

Inteligencia Artificial

Técnicas, métodos y aplicaciones

José T. Palma Méndez y Roque Marín Morales



**Mc
Graw
Hill**

INTELIGENCIA ARTIFICIAL

Métodos, técnicas y aplicaciones

INTELIGENCIA ARTIFICIAL

Métodos, técnicas y aplicaciones

COORDINADORES:
JOSÉ TOMÁS PALMA MÉNDEZ
ROQUE MARÍN MORALES

Universidad de Murcia



MADRID * BOGOTÁ * BUENOS AIRES * CARACAS * GUATEMALA * LISBOA * MÉXICO *
NUEVA YORK * PANAMÁ * SAN JUAN * SANTIAGO * SAO PAULO
AUCKLAND * HAMBURGO * LONDRES * MILÁN * MONTREAL * NUEVA DELHI * PARÍS
SAN FRANCISCO * SIDNEY * SINGAPUR * ST. LOUIS * TOKIO * TORONTO

Inteligencia Artificial: Métodos, técnicas y aplicaciones

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

DERECHOS RESERVADOS ©2008, respecto a la primera edición en español, por
McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

Edificio Valrealty, 1.ª Planta
Basauri, 17
28023 Aravaca (Madrid)

ISBN: 978-84-481-5618-3
Depósito legal: M.

Editor: José Luis García Jurado
Técnico editorial: Blanca Pecharromán Narro
Compuesto en: DIIC. Universidad de Murcia
Impreso en:

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Lista de Autores

Carlos Alonso González

Grupo de Sistemas Inteligentes
Departamento de Informática
Universidad de Valladolid

Federico Barber Sanchís

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Alberto José Bugarín Díz

Grupo de Sistemas Inteligentes
Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

José Joaquín Cañas Martínez

Grupo de Ingeniería de Datos, del Conocimiento y del Software
Departamento de Lenguajes y Computación
Universidad de Almería

Óscar Corcho

Grupo de Ingeniería Ontológica
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid

Miguel Delgado Calvo-Flores

Grupo de Razonamiento Aproximado e Inteligencia Artificial
Departamento de Ciencias de la Computación e IA
Universidad de Granada

Paulo Félix Lamas

Grupo de Sistemas Inteligentes
Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

Mariano Fernández López

Escuela Politécnica Superior
Ingeniería del Software y del Conocimiento
Universidad San Pablo CEU Madrid

César Ferri Ramírez

Grupo de Extensiones de la Programación Lógica
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Óscar Fontela Romero

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Computación
Universidad de A Coruña

Antonio Garrido Tejero

Grupo de Razonamiento en Planificación y Scheduling
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Asunción Gómez Pérez

Grupo de Ingeniería Ontológica
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid

Bertha Guijarro Berdiñas

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Computación
Universidad de A Coruña

Luis Daniel Hernández Molinero

Grupo de Sistemas Inteligentes
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

José Hernández Orallo

Grupo de Extensiones de la Programación Lógica
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Mario Hernández Tejera

Grupo de Inteligencia Artificial y Sistemas
Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria.

Fernando Jiménez Barrionuevo

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

José Manuel Juárez Herrero

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

Javier Lorenzo Navarro

Grupo de Inteligencia Artificial y Sistemas
Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria

Adolfo Lozano Tello

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de Sistemas Informáticos y Telemáticos
Universidad de Extremadura

Roque Marín Morales

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

Juan Méndez Rodríguez

Grupo de Inteligencia Artificial y Sistemas
Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria.

José Mira Mira

Departamento de Inteligencia Artificial
ETSI Informática
UNED

Vicente Moret Bonillo

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Computación
Universidad de A Coruña

Eduardo Mosqueira Rey

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Computación
Universidad de A Coruña

Eva Onaindia Rivaherrera

Grupo de Razonamiento en Planificación y Scheduling
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

José Tomás Palma Méndez

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

Juan Pavón Mestras

Grupo de Agentes Software: Ingeniería y Aplicaciones
Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid

María José Ramírez Quintana

Grupo de Extensiones de la Programación Lógica
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

María Camino Rodríguez Vela

Centro de Inteligencia Artificial
Departamento de Informática
Universidad de Oviedo

Miguel Ángel Salido Gregorio

Grupo de Tecnología Informática e Inteligencia Artificial
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Noelia Sánchez Meroño

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Computación
Universidad de A Coruña

Guido Sciavicco

Grupo de Inteligencia Artificial e Ingeniería del Conocimiento
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

María del Carmen Suárez de Figueroa Baonza

Grupo de Ingeniería Ontológica
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid

María Jesús Taboada Iglesias

Grupo de Ingeniería del Conocimiento aplicada a la Medicina
Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

Ramiro Valera Arias

Centro de Inteligencia Artificial
Departamento de Informática
Universidad de Oviedo

Amparo Vila Miranda

Grupo de Razonamiento Aproximado e Inteligencia Artificial
Departamento de Ciencias de la Computación e IA
Universidad de Granada

Índice general

I INTRODUCCIÓN	1
1 Aspectos conceptuales de la IA y la IC	3
<i>José Mira Mira</i>	
1.1 Introducción	3
1.2 La IA como Ciencia y como IC	6
1.2.1 IA como Ciencia	7
1.2.2 IA como Ingeniería	7
1.3 Perspectiva histórica: fundamentos y metodología	10
1.4 Paradigmas actuales en IA	12
1.4.1 El paradigma simbólico	14
1.4.2 El paradigma situado	16
1.4.3 El paradigma conexiónista	18
1.4.4 El paradigma híbrido	20
1.5 El conocer humano y el conocer de las máquinas	23
1.6 Algunas sugerencias	26
1.7 Resumen	27
Referencias	28
II REPRESENTACIÓN DE CONOCIMIENTO E INFERENCIA	31
2 Lógica y representación del conocimiento	33
<i>Guido Sciavicco</i>	
2.1 Introducción: ¿Por qué la Lógica?	33
2.2 Lógica proposicional	36
2.2.1 Sintaxis y semántica	36
2.2.2 Poder expresivo y límites de la Lógica Proposicional	40
2.2.3 Métodos deductivos semánticos y coste computacional	41
2.3 Lógica de primer orden	46
2.3.1 Sintaxis y semántica	46

2.3.2	Poder expresivo y límites de la lógica de primer orden	49
2.3.3	Métodos deductivos y coste computacional	49
2.3.4	Lógicas de orden superior al primero	54
2.3.5	Fragmentos de LPO	56
2.4	Extensiones de las lógicas clásicas	57
2.4.1	¿Por qué extender las lógicas clásicas?	57
2.4.2	Lógicas no monotónicas, razonamiento del sentido común y otras consideraciones	57
2.4.3	Lógicas modales y mundos posibles	59
2.4.4	Métodos deductivos y coste computacional de la Lógica Modal	61
2.5	Aplicaciones: el ejemplo de las lógicas temporales	64
2.5.1	Tipos de lógicas temporales	64
2.5.2	Lógicas temporales basadas en puntos	64
2.5.3	Lógicas temporales basadas en intervalos	68
2.6	Ejercicios resueltos	71
2.7	Ejercicios propuestos	76
	Referencias	80
3	Sistemas basados en reglas	83
	<i>María Jesús Taboada Iglesias y Asunción Gómez Pérez</i>	
3.1	Introducción	83
3.2	Componentes básicos de los SBR	84
3.2.1	Base de Hechos	84
3.2.2	Base de Conocimiento	85
3.2.3	Motor de inferencias	87
3.3	Inferencia	88
3.3.1	Encadenamiento hacia delante	89
3.3.2	Encadenamiento hacia atrás	97
3.3.3	Reversibilidad	102
3.4	Técnicas de equiparación	104
3.4.1	Equiparación con variables	104
3.4.2	El algoritmo RETE	109
3.5	Técnicas de resolución de conflictos	111
3.6	Ventajas e inconvenientes	116
3.7	Dominios de aplicación	118
3.8	Resumen	119
3.9	Ejercicios resueltos	120
3.10	Ejercicios propuestos	127
	Referencias	130
4	Redes semánticas y marcos	131
	<i>María del Carmen Suárez de Figueroa Baonza y Asunción Gómez Pérez</i>	
4.1	Introducción	131
4.2	Redes semánticas	133

4.2.1	Representación de conocimiento	133
4.2.2	Representación de predicados no binarios	136
4.2.3	Representación de acciones	136
4.2.4	Representación de conocimiento disjunto	138
4.3	Inferencia de conocimiento en redes semánticas	139
4.3.1	Equiparación	139
4.3.2	Herencia de propiedades	140
4.4	Marcos	143
4.4.1	Representación de conocimiento	143
4.4.2	Criterios de diseño	155
4.5	Inferencia de conocimiento en SBM	157
4.5.1	Equiparación	157
4.5.2	Herencia de propiedades	159
4.5.3	Valores activos	161
4.6	Resumen	162
4.7	Ejercicios resueltos	163
4.8	Ejercicios propuestos	164
	Referencias	170
5	Ontologías	171
	<i>Asunción Gómez Pérez, Mariano Fernández López y Óscar Corcho</i>	
5.1	Introducción	171
5.2	Ontologías: definición y componentes	172
5.3	Una metodología para el desarrollo de ontologías: METHONTOLOGY	174
5.3.1	Proceso de desarrollo de ontologías y su ciclo de vida	174
5.3.2	Conceptualización de una ontología de entidades legales	178
5.4	Cómo construir una ontología legal con WebODE	188
5.5	Otros métodos y herramientas para desarrollar ontologías	190
5.5.1	Metodologías y métodos	190
5.5.2	Herramientas	191
5.6	Lenguajes de implementación de ontologías	194
5.7	Resumen	196
5.8	Ejercicios resueltos	197
5.9	Ejercicios propuestos	198
	Referencias	200
6	Sistemas basados en modelos probabilísticos	207
	<i>Luis Daniel Hernández Molinero</i>	
6.1	Introducción	207
6.2	Conceptos básicos de la teoría de la probabilidad	209
6.2.1	Fundamentos del cálculo de probabilidades	209
6.2.2	Interpretaciones de la probabilidad	211
6.2.3	Variables y probabilidades	214
6.2.4	Probabilidad condicionada e independencia	220
6.2.5	Regla de Bayes	222

6.3	Una introducción a las redes bayesianas	225
6.3.1	Redes causales y d-separación	225
6.3.2	Redes bayesianas	228
6.4	Razonamiento con redes bayesianas	234
6.4.1	Inferencia probabilística	236
6.4.2	Interpretación de evidencias	249
6.5	Referencias bibliográficas y software	253
6.6	Ejercicios propuestos	254
	Referencias	259
7	Conjuntos borrosos	263
	<i>Paulo Félix Lamas y Alberto José Bugarín Diz</i>	
7.1	Introducción	263
7.2	Conjuntos borrosos	264
7.3	Semántica de los conjuntos borrosos	266
7.4	Teorías de conjuntos borrosos	268
7.5	Variable lingüística	272
7.6	Principio de extensión	274
7.6.1	Aritmética borrosa	275
7.7	Relaciones borrosas	277
7.7.1	Relaciones de similitud	278
7.7.2	Relaciones de comparación	279
7.7.3	Proyección. Extensión cilíndrica	280
7.7.4	Composición de relaciones	281
7.8	El condicional	284
7.9	Cualificación lingüística	287
7.10	Razonamiento borroso	289
7.10.1	Condicionales con antecedente múltiple	292
7.11	Cuantificación	293
7.12	Lecturas recomendadas	298
7.13	Resumen	298
7.14	Ejercicios resueltos	299
7.15	Ejercicios propuestos	304
	Referencias	305
III	TÉCNICAS	307
8	Introducción a las técnicas de búsqueda	309
	<i>María Camino Rodríguez Vela y Ramiro Varela Arias</i>	
8.1	Introducción	309
8.2	Algunos ejemplos	310
8.2.1	Generación de planes de actuación de robots	310
8.2.2	Problemas de rutas óptimas en grafos	312
8.2.3	Juegos con contrincante	313

8.3	Formulación de problemas de búsqueda	314
8.4	Métodos de búsqueda sin información	315
8.4.1	Recorrido de árboles	316
8.4.2	Recorrido de grafos	321
8.5	Resumen	326
8.6	Ejercicios resueltos	327
8.7	Ejercicios propuestos	335
	Referencias	337
9	Técnicas basadas en búsquedas heurísticas	339
	<i>María Camino Rodríguez Vela y Ramiro Varela Arias</i>	
9.1	Introducción	339
9.2	Búsqueda primero el mejor	343
9.3	El algoritmo A*	343
9.3.1	Descripción del algoritmo A*	344
9.3.2	Propiedades formales	345
9.3.3	Diseño de heurísticos simples	353
9.3.4	Relajación de las condiciones de optimalidad	357
9.3.5	Diseño sistemático de heurísticos	360
9.4	Búsqueda con memoria limitada	362
9.4.1	Algoritmo IDA*	362
9.4.2	Algoritmo SMA*	364
9.5	Algoritmos voraces	366
9.6	Algoritmos de ramificación y poda	366
9.7	Algoritmos de mejora iterativa o búsqueda local	369
9.7.1	Algoritmos de escalada o máximo gradiente	371
9.7.2	Temple simulado	372
9.7.3	Búsqueda tabú	373
9.8	Resumen	374
9.9	Ejercicios resueltos	375
9.10	Ejercicios propuestos	381
	Referencias	383
10	Problemas de satisfacción de restricciones (CSP)	385
	<i>Federico Barber Sanchís y Miguel Ángel Salido Gregorio</i>	
10.1	Introducción	385
10.2	Definiciones y conceptos básicos	386
10.2.1	Definición de un problema de satisfacción de restricciones	387
10.2.2	Definición y tipología de las restricciones	388
10.3	Ejemplos de CSP y su modelización	389
10.3.1	Coloración del mapa	389
10.3.2	Criptografía	390
10.3.3	El problema de las N -reinas	391

10.4	Técnicas CSP	392
10.4.1	Métodos de búsqueda	392
10.4.2	Técnicas de inferencia	394
10.4.3	Técnicas híbridas	402
10.5	Heurísticas de búsqueda	407
10.5.1	Heurísticas de ordenación de variables	408
10.5.2	Ordenación de valores. Tipos	412
10.6	Extensiones de CSP	413
10.6.1	CSP no binarios	414
10.6.2	CSP distribuidos (DisCSP)	416
10.6.3	CSP temporales y CSP dinámicos	418
10.6.4	Otras extensiones	421
10.7	Lecturas recomendadas	423
10.8	Resumen	423
10.9	Ejercicios resueltos	424
10.10	Ejercicios propuestos	427
	Referencias	431
11	Computación Evolutiva	433
	<i>Fernando Jiménez Barrionuevo</i>	
11.1	Introducción	433
11.2	Un algoritmo genético simple	436
11.2.1	Representación	436
11.2.2	Obtención de la población inicial	437
11.2.3	Función de evaluación	437
11.2.4	Selección, muestreo, operadores genéticos y sustitución generacional	438
11.2.5	Parámetros de entrada	440
11.2.6	Ejemplo: Optimización de una función simple	441
11.3	Fundamentos de los algoritmos genéticos	443
11.4	Diseño de algoritmos evolutivos	445
11.4.1	Representación	446
11.4.2	Esquemas de selección, muestreo y sustitución generacional	446
11.4.3	Operadores de variación	448
11.4.4	Manejo de restricciones	449
11.4.5	Optimización multiobjetivo	451
11.4.6	Evaluación y validación de algoritmos evolutivos	456
11.5	Lecturas recomendadas	457
11.6	Resumen	458
11.7	Ejercicios resueltos	458
11.8	Ejercicios propuestos	466
	Referencias	468

IV TAREAS	471
12 Diagnosis	473
<i>Carlos Alonso González</i>	
12.1 Introducción	473
12.1.1 Algunas definiciones	474
12.2 Elementos básicos de un sistema de diagnosis	477
12.2.1 Espacio de búsqueda	477
12.2.2 Modelo del sistema	478
12.2.3 Ejemplo de sistema a diagnosticar	478
12.2.4 Operaciones o subtareas	479
12.3 Diagnosis basada en árboles de fallos	485
12.4 Diagnosis basada en modelos de clasificación simbólica	489
12.4.1 Clasificación simple	489
12.4.2 Clasificación jerárquica	497
12.4.3 Otros modelos de clasificación simbólica	502
12.4.4 Implementación de un modelo de clasificación simbólica	503
12.4.5 Un sistema de diagnosis clásico: MYCIN	505
12.5 Diagnosis basada en modelos: la aproximación basada en consistencia	508
12.5.1 Motivación	508
12.5.2 Diagnosis basada en consistencia	510
12.5.3 Modelo formal de la diagnosis basada en consistencia	513
12.5.4 Limitaciones de la diagnosis basada en consistencia	518
12.5.5 Paradigma computacional: General Diagnostic Engine	519
12.6 Métodos y modelos para la diagnosis	523
12.6.1 Métodos de clasificación	524
12.6.2 Métodos basados en casos	525
12.6.3 Métodos basados en modelos	526
12.6.4 Origen de los modelos	526
12.7 Resumen	528
12.8 Lecturas recomendadas	529
12.9 Ejercicios Propuestos	530
Referencias	534
13 Planificación	537
<i>Eva Onaindia de la Rivaherrera y Antonio Garrido Tejero</i>	
13.1 Introducción	537
13.2 Problema de planificación	539
13.3 Lenguaje de planificación PDDL	541
13.4 Planificación en un espacio de estados	543
13.4.1 Búsqueda hacia delante	543
13.4.2 Búsqueda hacia atrás	545
13.5 Planificación de orden parcial	548
13.5.1 Estructura de un plan de orden parcial	549
13.5.2 Búsqueda en un espacio de planes para POP	551

13.5.3	Heurísticas para planificación de orden parcial	555
13.6	Planificación basada en grafos de planificación	557
13.6.1	Grafos de planificación	557
13.6.2	Extracción de planes: Graphplan	560
13.6.3	Heurísticas basadas en grafos de planificación	564
13.7	Planificación basada en satisfacibilidad	565
13.8	Planificación para el mundo real	566
13.8.1	Planificación numérica: tiempo + recursos	568
13.8.2	Planificación jerárquica	571
13.8.3	Planificación con incertidumbre	572
13.9	Lecturas recomendadas	573
13.10	Ejercicios resueltos	575
13.11	Ejercicios propuestos	579
	Referencias	582
14	Control	589
<i>José M. Juárez Herrero, José J. Cañadas Martínez y Roque Marín Morales</i>		
14.1	Introducción	589
14.2	Conceptos fundamentales	590
14.2.1	Problema de control	590
14.2.2	Componentes fundamentales de un sistema de control	591
14.2.3	Tipos de sistemas de control	591
14.2.4	Soluciones convencionales	593
14.3	Métodos genéricos para control	597
14.3.1	Control convencional frente a control inteligente	597
14.3.2	Tarea genérica para el Control	599
14.4	Control borroso	601
14.4.1	Estructura básica	602
14.4.2	Variables LHS / RHS	605
14.4.3	Obtención de entradas borrosas: fuzzificación	606
14.4.4	Base de Reglas Borrosas	606
14.4.5	Mecanismo de inferencias	607
14.4.6	Obtención de salidas precisas: defuzzificación	612
14.5	Arquitecturas y herramientas	614
14.5.1	Herramientas para control borroso	614
14.5.2	Ejemplo de desarrollo de un controlador borroso	616
14.6	Arquitecturas de pizarra	624
14.6.1	Metáfora de la pizarra	624
14.6.2	Componentes básicos de un sistema de pizarra	625
14.6.3	Ciclo de control	626
14.6.4	Características de las arquitecturas de pizarra	630
14.7	Sistemas en tiempo real basados en conocimiento	632
14.7.1	Sistemas en tiempo real	632
14.7.2	Concepto de sistema en tiempo real basado en conocimiento .	633
14.7.3	Características de los STRBC	633

14.7.4	Eventos asíncronos	634
14.7.5	Razonamiento no monótono	634
14.7.6	Modo de operación continuado	634
14.7.7	Datos inciertos o imprecisos	635
14.7.8	Razonamiento temporal	635
14.7.9	Tiempo de respuesta garantizado	636
14.7.10	Foco de atención	637
14.7.11	Herramientas para Sistemas Expertos en Tiempo Real	637
14.8	Lecturas recomendadas	639
14.9	Resumen	640
14.10	Ejercicios propuestos	641
	Referencias	644
V	APRENDIZAJE Y MINERÍA DE DATOS	647
15	Redes neuronales	649
<i>Bertha Guijarro Berdiñas, Óscar Fontela Romero y Noelia Sánchez Meroño</i>		
15.1	Introducción	649
15.1.1	¿Qué es una red de neuronas?	649
15.1.2	Tipos básicos de problemas	651
15.1.3	Proceso de entrenamiento o aprendizaje	652
15.2	Métodos de aprendizaje supervisado	654
15.2.1	Redes de neuronas de una capa: el Perceptrón	654
15.2.2	El perceptrón multicapa	657
15.2.3	Redes de base radial	664
15.3	Métodos de aprendizaje no supervisados	672
15.3.1	Mapas autoorganizativos	673
15.4	Máquina de Vectores Soporte	674
15.4.1	Caso linealmente separable	675
15.4.2	Caso no linealmente separable	679
15.4.3	Máquinas de vectores soporte no lineales	681
15.4.4	Máquinas de Vectores Soporte Multiclasé	682
15.5	Resumen	683
15.6	Ejercicios propuestos	684
	Referencias	687
16	Técnicas de agrupamiento	691
<i>Amparo Vila Miranda y Miguel Delgado Calvo-Flores</i>		
16.1	Introducción	691
16.2	Conceptos básicos	692
16.3	Los datos de partida	695
16.3.1	La matriz de elementos	696
16.3.2	Índices de proximidad: distancias y semejanzas	696
16.4	Técnicas de agrupamiento jerárquico	699

16.4.1 Ideas básicas	699
16.4.2 Algoritmos para el agrupamiento jerárquico	700
16.5 Técnicas de agrupamiento particional	703
16.5.1 Ideas iniciales	703
16.5.2 El método de las k -medias	704
16.5.3 DBSCAN: un método basado en el análisis de densidad	707
16.6 Nuevos resultados y extensiones	709
16.6.1 Resultados recientes sobre agrupamiento jerárquico	709
16.6.2 Extensiones a los métodos particionales prototípicos: los métodos de k-medoides	711
16.7 Técnicas de agrupamiento borroso	713
16.7.1 Agrupamientos borrosos particionales	714
16.7.2 Agrupamientos jerárquicos y conjuntos borrosos	717
16.8 Herramientas software para el agrupamiento	718
16.9 Lecturas recomendadas	720
16.10 Resumen	721
16.11 Ejercicios propuestos	721
Referencias	722
17 Aprendizaje de árboles y reglas de decisión	725
<i>César Ferri Ramírez y María José Ramírez Quintana</i>	
17.1 Introducción	725
17.2 Inducción de árboles de decisión	727
17.3 Aprendizaje de reglas por cobertura	732
17.4 Reestructuración de reglas	736
17.5 Otras aplicaciones de los árboles de decisión	739
17.6 Extensiones de árboles y reglas de decisión	741
17.6.1 Métodos multiclasicadores	741
17.7 Sistemas y aplicabilidad	743
17.8 Lecturas recomendadas	746
17.9 Resumen	747
17.10 Ejercicios resueltos	748
17.11 Ejercicios propuestos	755
Referencias	758
18 Técnicas de extracción de reglas	763
<i>Maria José Ramírez Quintana y José Hernández Orallo</i>	
18.1 Introducción	763
18.2 Técnicas de extracción de reglas a partir de modelos de caja negra	766
18.3 Técnicas de extracción de reglas sobre redes neuronales	770
18.3.1 Métodos globales	772
18.3.2 Métodos locales	776
18.4 Técnicas de extracción de reglas sobre otros paradigmas	779
18.5 Técnicas de extracción de reglas borrosas	780
18.5.1 Técnicas de extracción directa (caja negra)	781

18.5.2	Sistemas neuroborrosos (neuro-fuzzy)	784
18.6	Lecturas recomendadas	786
18.7	Resumen	786
18.8	Ejercicios resueltos	788
18.9	Ejercicios propuestos	790
	Referencias	792
VI	ASPECTOS METODOLÓGICOS Y APLICACIONES	799
19	Ingeniería del Conocimiento	801
	<i>Adolfo Lozano Tello</i>	
19.1	Introducción a la Ingeniería del Conocimiento	801
19.2	Adquisición del conocimiento	803
19.2.1	Técnicas manuales de adquisición del conocimiento	804
19.2.2	Técnicas semiautomáticas de adquisición del conocimiento	806
19.2.3	Técnicas automáticas de adquisición del conocimiento	808
19.2.4	Adquisición del conocimiento a partir de un grupo de expertos	809
19.3	Sistemas basados en conocimiento	810
19.3.1	Estructura de los SBCs	811
19.3.2	Propiedades de los SBCs	812
19.4	Métodos de desarrollo de sistemas basados en conocimiento	813
19.4.1	La metodología CommonKADS	815
19.5	Construcción de SBC usando CK	816
19.5.1	Modelado del contexto en CommonKADS	818
19.5.2	Modelado conceptual en CommonKADS	829
19.5.3	Modelado artefactual en CommonKADS	845
19.6	Lecturas recomendadas	851
19.7	Resumen	851
19.8	Ejercicio propuesto	852
	Referencias	853
20	Sistemas multiagentes	857
	<i>Juan Pavón Mestras</i>	
20.1	Introducción	857
20.2	Arquitecturas de agentes	859
20.2.1	Agentes deliberativos	861
20.2.2	Agentes reactivos	864
20.2.3	Agentes híbridos	866
20.3	Sociedades de agentes	867
20.3.1	Organizaciones de agentes	868
20.3.2	Coordinación en SMA	869
20.3.3	Negociación, confianza y reputación	869
20.4	Comunicación entre agentes	870
20.4.1	El lenguaje FIPA-ACL	872

20.4.2	Protocolos de comunicación	874
20.5	Métodos, herramientas y plataformas	875
20.5.1	Programación de agentes	877
20.5.2	Plataformas de agentes FIPA	878
20.6	Aplicaciones de los agentes	880
20.7	Ejercicios resueltos: Análisis y diseño de un SMA	882
	Ejercicios Resueltos: Análisis y diseño de un SMA	882
20.8	Ejercicios propuestos	888
	Referencias	889
21	Verificación y validación de sistemas inteligentes	891
	<i>Vicente Moret Bonillo y Eduardo Mosqueira Rey</i>	
21.1	Introducción	891
21.2	Verificación de sistemas inteligentes	893
21.2.1	Cumplimiento de las especificaciones	893
21.2.2	Verificación de los mecanismos de inferencia	894
21.2.3	Verificación de la base de conocimientos	895
21.2.4	Influencia de las medidas de incertidumbre	898
21.3	Validación de sistemas inteligentes	899
21.3.1	Personal involucrado	899
21.3.2	Qué validar	900
21.3.3	Casuística de validación	900
21.3.4	Validación contra el experto	902
21.3.5	Validación contra el problema	903
21.4	Métodos cuantitativos de validación	904
21.4.1	Medidas de pares	904
21.4.2	Medidas de grupo	910
21.4.3	Ratios de acuerdo	913
21.5	Síntesis metodológica del proceso de validación	918
21.5.1	Planificación del proceso	918
21.5.2	Fase de aplicación de técnicas	920
21.5.3	Interpretación de resultados	921
21.6	Resumen	922
21.7	Ejercicios resueltos	923
21.8	Ejercicios propuestos	931
	Referencias	934
22	Razonamiento basado en casos	937
	<i>José Manuel Juárez Herrero y José Tomás Palma Méndez</i>	
22.1	Introducción	937
22.2	Sistemas de Razonamiento Basado en Casos	939
22.3	Elementos de un SRBC	941
22.3.1	Los casos y su descripción	942
22.3.2	Librería de casos	944
22.3.3	Determinación de casos similares	947

22.4	El ciclo RBC. Etapas-RE	953
22.4.1	Recuperar	954
22.4.2	Reutilizar	956
22.4.3	Revisar	957
22.4.4	Retener	958
22.4.5	Mantenimiento	958
22.5	Aplicaciones de los sistemas RBC	959
22.6	Herramientas para el desarrollo de SRBC	962
22.7	Lecturas recomendadas y recursos en Internet	966
22.8	Resumen	967
22.9	Ejercicios resueltos	968
22.10	Ejercicios propuestos	972
	Referencias	973
23	Reconocimiento de Formas	975
	<i>Mario Hernández Tejera, Javier Lorenzo Navarro y Juan Méndez Rodríguez</i>	
23.1	Introducción	975
23.2	Modelos estadísticos	979
23.2.1	Clasificador bayesiano de mínimo error	980
23.2.2	Clasificador bayesiano de mínimo riesgo	982
23.2.3	Distribuciones normales multivariantes	984
23.2.4	Estimación de distribuciones	985
23.3	Modelos geométricos de la decisión	987
23.3.1	Funciones discriminantes	987
23.3.2	Clasificación por funciones de distancia	989
23.3.3	Separabilidad Lineal y la dimensión Vapnik-Chervonenkis	993
23.4	Aprendizaje de clasificadores lineales	995
23.4.1	Procedimiento perceptrón	996
23.4.2	Procedimientos de mínimo error cuadrático	998
23.4.3	Máquina de vectores soportes	1001
23.5	Aprendizaje de clasificadores no lineales	1005
23.5.1	Topología de perceptrones multicapa	1005
23.5.2	Redes de funciones de base radial	1007
23.6	Selección de características	1009
23.6.1	Análisis de componentes principales	1010
23.6.2	Análisis de componentes independientes	1011
23.6.3	Escalado multidimensional	1013
23.7	Validación y comparación de clasificadores	1014
23.7.1	Test de clasificadores	1015
23.7.2	Comparación de prestaciones	1017
23.8	Ejercicios propuestos	1018
	Referencias	1022

Prólogo

Una definición comúnmente aceptada relaciona la disciplina de la Inteligencia Artificial (IA) con el análisis y el diseño de sistemas artificiales autónomos capaces de exhibir un comportamiento inteligente. Se asume que, para que un agente actúe intelligentemente, debe poder percibir su entorno, elegir y planificar sus objetivos, actuar hacia la consecución de estos objetivos aplicando algún principio de racionalidad e interactuar con otros agentes inteligentes, sean estos artificiales o humanos.

Esta disciplina no es nueva. Hace algo más de cincuenta años, el 31 de Agosto de 1955, Marvin Minsky, John McCarty, Nathan Rochester y Claude Shannon propusieron la celebración de una reunión de dos meses de duración, que tuvo lugar en el Dartmouth College durante el verano de 1956. Su principal objetivo era discutir: ¿la conjectura de que todos los aspectos del aprendizaje o de cualquier otra característica de la inteligencia pueden, en principio, ser descritos de modo tan preciso que se pueda construir una máquina capaz de simularlos? El tema parecía tan novedoso que acuñaron un nuevo término para él: Inteligencia Artificial.

Se pensaba entonces que éste era un objetivo factible, que se alcanzaría en unas pocas décadas. Las previsiones eran, ciertamente, demasiado optimistas. Medio siglo después, podemos hacer un balance de los logros conseguidos. Estamos aún lejos de ese objetivo genérico. No existen aún agentes perfectamente autónomos, con una inteligencia tan completa y tan compleja como la de los seres humanos. Sin embargo, disponemos ahora de una colección extensa y diversa de modelos y técnicas que nos han permitido construir dispositivos útiles, aunque imperfectos, que facilitan la resolución de tareas de alta complejidad, inabordables mediante otras técnicas convencionales de la Informática.

El enfoque de la IA también ha cambiado. Unos años antes de la reunión de Dartmouth, en 1952, Ashby proponía el objetivo, ambicioso también, pero más realista, de construir dispositivos que actuasen como “amplificadores de la inteligencia” humana. A lo largo del medio siglo transcurrido desde entonces, el campo de la IA ha ido evolucionando de forma natural hacia esa antigua propuesta de Ashby. Aunque no es el único enfoque aceptado, hoy predomina la idea de que la implementación de sistemas convencionales con inteligencia embebida, es un objetivo más práctico, más útil y más factible que la búsqueda de la inteligencia general en un único dispositivo. Hoy proliferan los asistentes inteligentes que, embebidos en aplicaciones de todo tipo, desde sistemas de información hasta procesadores de textos, ayudan al usuario en la

realización de tareas complejas y proporcionan un valor añadido a los sistemas convencionales. Se trata, por tanto, de conseguir una sinergia efectiva entre inteligencia e información.

En los últimos años, y con la revolución de las comunicaciones, esta sinergia se ha extendido hacia una ecuación más completa: “Información + Inteligencia + Ubicuidad”. Por poner unos ejemplos, los sistemas de inteligencia ambiental, sensibles al contexto en el que se mueven, el trabajo hacia una web semántica o los dispositivos para la telemonitorización inteligente de parámetros biológicos, muestran de forma patente hacia donde está evolucionado el campo de la IA.

Sea como sea, estos cincuenta años de investigación en IA, no han sido en absoluto improductivos. De hecho, la IA se ha convertido en unas de las áreas más prolíficas en cuanto a resultados de investigación. No podemos obviar que muchos de los desarrollos que surgieron de grupos de investigación en IA están presentes en muchas de las actividades que comúnmente desarrollamos, desde los sistemas de navegación de los automóviles, hasta los pequeños controladores inteligentes de algunos electrodomésticos. Tampoco podemos obviar los avances que se han producido en campos como la robótica que han supuesto una revolución en los métodos de fabricación. Toda esta labor investigadora ha producido gran cantidad de técnicas y metodologías que han dado lugar a numerosas disciplinas dentro de la IA, que dibujan un panorama que resulta casi imposible de abordar en un sólo libro.

La visión de la IA como ingeniería ha predominado sobre la visión de la IA como ciencia, y hoy en día disponemos, por tanto, de una amplia variedad de modelos, metodologías, técnicas y aplicaciones. El peso de la IA dentro del área de la Informática la convirtió hace tiempo, no sólo en objeto de investigación, sino también en un objetivo docente de primer orden dentro de cualquier plan de estudios medianamente completo.

Este libro pretende proporcionar una contribución sólida, y razonablemente completa, a la docencia de la IA en los planes de estudio de la Inginería Informática. En Inteligencia Artificial: Métodos, Técnicas y Aplicaciones, se han intentado conjugar los tópicos clásicos de la IA, que se vienen cubriendo en la docencia de grado, con otros tópicos avanzados que pueden ser ubicados tanto en la docencia de grado como de posgrado.

De esta forma, el libro se ha dividido en seis partes. El libro comienza con una sección de Introducción, donde se analizan los diferentes paradigmas en los que se basa la IA en la actualidad y se proporciona una visión crítica de los retos y los logros en este campo. Es importante emplazar a los alumnos, ya desde el principio, a abordar el estudio de la IA con un espíritu crítico que les permita situar en su justa medida los contenidos concretos que van a estudiar después. Con el paso de los años, el campo de la IA ha ido siendo invadido por una terminología que, a menudo, es más ampulosa que descriptiva, empezando por el propio nombre de la disciplina: Inteligencia Artificial. Por consiguiente, el objetivo de la primera parte del libro es dibujar un marco general en una escala que, los editores y el autor del primer capítulo, entendemos es la correcta.

En la segunda parte, denominada Representación de Conocimiento e Inferencia, se abordan desde los aspectos clásicos de representación de conocimiento y técnicas

clásicas de razonamiento, hasta aspectos más avanzados relacionados con la gestión de la incertidumbre, para de esta forma dar una visión lo más completa posible de la evolución que han sufrido los mecanismos para representar conocimiento y razonar sobre él. La tercera parte describe un conjunto de Técnicas básicas de la IA que se utilizan en la resolución de gran cantidad de problemas, entre las que se incluyen las clásicas técnicas de búsqueda hasta aspectos más avanzados como la computación evolutiva y satisfacción de restricciones. En la cuarta parte se presentan un conjunto de procesos complejos de resolución de problemas, que hemos denominado Tareas, que se pueden aplicar en la resolución de determinadas clases de problemas, como pueden ser la planificación y el diagnóstico. La quinta parte está centrada en los aspectos relacionados con el Aprendizaje y la Minería de datos, donde se hace una introducción a algunas técnicas de aprendizaje cubriendo aspectos relacionados con el aprendizaje supervisado y no supervisado, así como técnicas de aprendizaje no simbólicas. Finalmente, la sexta parte trata los Aspectos Metodológicos y Aplicaciones, y en ella se ha intentado abordar temas relacionados con el análisis y desarrollo de sistemas inteligentes como la Ingeniería del Conocimiento, los Sistemas Multiagentes y la Verificación y Validación, así como algunas aplicaciones, resultado de combinación de distintas técnicas analizadas a lo largo del libro, como el Razonamiento Basado en Casos y el Reconocimiento de Formas.

Cada uno de los capítulos ha sido desarrollado por expertos de contrastada reputación en su campo, lo que ha permitido, no sólo exponer los aspectos básicos relacionados con el capítulo de una forma clara, sino que se ha podido dar una visión más global de cada tema indicando referencias bibliográficas con las que profundizar en cada uno de los tópicos.

Sirvan estas líneas para agradecer a todos los autores que han colaborado en la redacción y corrección de cada uno de los capítulos, ya que evidentemente, sin su colaboración desinteresada no hubiera sido posible que este proyecto viese la luz. Queremos agradecer la paciencia que han mostrado ante las instrucciones y requerimientos que les hemos hecho llegar y a las que han respondido con celeridad. Mención especial queremos hacer a la editorial McGraw-Hill, que nos ha prestado todo su apoyo en el proceso de elaboración de este libro, y a los profesores Abraham Rodríguez Rodríguez, del Departamento de Informática y Sistemas de la Universidad de Las Palmas de Gran Canaria, y José del Sagrado Martínez, del departamento de Lenguajes y Computación de la Universidad de Almería, por su colaboración en la corrección de varios capítulos de esta obra.

José Tomás Palma Méndez
Roque Marín Morales
Coordinadores

Parte I

INTRODUCCIÓN

Capítulo 1

Aspectos conceptuales de la Inteligencia Artificial y la Ingeniería del Conocimiento

José Mira Mira

Universidad Nacional de Educación a Distancia

1.1 Introducción

Actualmente se acepta, con alto grado de consenso entre los profesionales del campo de la computación, que el propósito general de la IA es desarrollar: (1) Modelos conceptuales, (2) procedimientos de reescritura formal de esos modelos y (3) estrategias de programación y máquinas físicas para reproducir de la forma más eficiente y completa posible las tareas cognitivas y científico-técnicas más genuinas de los sistemas biológicos a los que hemos etiquetado de inteligentes [Mira y Delgado, 1995b]. Esto nos obliga a aceptar ya desde el comienzo del capítulo que el avance de la IA está necesariamente limitado por los avances en las técnicas de modelado, formalización y programación y por la evolución en los materiales y las arquitecturas de los computadores y los dispositivos electromecánicos (“robots”) en los que se instala el cálculo.

Quedémonos inicialmente con la idea de que los cuatro grandes objetivos de la IA son *modelar, formalizar, programar e implementar* máquinas soporte capaces de interactuar de forma no trivial con el medio. Esto también lo hace la computación convencional y para aclarar la diferencia recurrimos a Leibniz, quien nos decía que “todo lo que sepamos describir de forma *clara, completa, precisa e inequívoca* es computable”. Por eso el conjunto de tareas y métodos propios de la IA son todas aquellas y aquellos para los que en la actualidad sólo disponemos de descripciones poco claras, incompletas, imprecisas y con alto grado de dudas y errores potenciales, debidos a su complejidad. Por ejemplo, la percepción, el razonamiento creativo, la comprensión y producción del lenguaje natural o los procesos de aprendizaje.

Esta aproximación a los objetivos de la IA conlleva la suposición subyacente de que efectivamente podemos sintetizar estas tareas cognitivas, de que podremos reducir el lenguaje natural a lenguajes formales, la semántica a la sintaxis, los símbolos neurofisiológicos a símbolos estáticos, el conocimiento a arquitecturas y, finalmente, el procesado biológico de la información a un cálculo residente en un circuito electrónico.

La gran hipótesis de la IA fue que en un número corto de años (10, 20, 30, 40, ahora ya 50) iba a ser posible sintetizar los procesos cognitivos y conseguir “*inteligencia general en máquinas*” (“every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it” [McCarthy y otros, 1955; Rich, 1990]). Es decir, que no existían impedimentos físicos, constitutivos ni formales para este objetivo y que sólo era cuestión de recursos. Cincuenta años después de la conferencia del Dartmouth College no todos los profesionales del campo estamos de acuerdo con esta afirmación, ni tampoco vemos que sea necesaria. No estamos de acuerdo primero por la propia naturaleza general y abstracta de la palabra inteligencia que se usa para describir una medida global de la calidad de todos los procesos cognitivos de un ser vivo y de su capacidad de adaptación a los cambios del medio en el que existen otros seres vivos de complejidad comparable. De hecho el término inteligencia es un concepto precientífico cuya utilidad actual es, cuando poco, discutible. Dentro de la propia psicología se propone [Gardner, 2004] descomponer el término y hablar de “inteligencias múltiples” o de “inteligencia colectiva” intentando poner de manifiesto en el primer caso la existencia de habilidades relativamente autónomas y en el segundo el carácter social y distribuido del conocimiento humano.

Cuando intentamos concretar más el significado del término inteligencia, para saber qué es lo que queremos mimetizar en una máquina, nos encontramos con el conjunto de conocimientos y habilidades a los que hace referencia toda la neurofisiología, la psicología, la sociología y la filosofía. Parece que se concreta algo al afirmar que la IA tiene que ver “sólo” con los conocimientos necesarios para la comprensión y la síntesis de procedimientos de solución de problemas complejos que exigen la capacidad de procesamiento simbólico y relacional. Pero ahora hay que especificar qué hace que un problema sea “complejo” y por qué su solución debe de exigir procesamiento simbólico. Dos de las formas posibles de contestar a esta pregunta han dado lugar a las dos aproximaciones dominantes en IA: a simbólica o representacional y la del conexionismo situado. La primera hace énfasis en la vía descendente y en el uso de conceptos del lenguaje natural (hechos y reglas) para representar el conocimiento necesario para resolver problemas de decisión que no necesitan un robot como componente imprescindible en su implementación. La segunda hace énfasis en la vía ascendente y en el uso de conceptos de más bajo nivel semántico. Considera la inteligencia como una forma “superior” de adaptación al medio y se apoya en conductas y mecanismos implementables en un robot real que tiene que interactuar con un entorno real concreto. Por otra parte, volviendo a la gran hipótesis de la época fundacional de la IA, tampoco creemos que sea necesaria la síntesis de inteligencia general en máquinas en el sentido de equivalencia total o “clonación”. Por el contrario, la IA como toda ciencia e ingeniería, debe de tener un carácter *instrumental* y sus objetivos no deben de ser

otros que (1) ayudar a comprender los procesos neurofisiológicos, cognitivos y sociales y (2) prolongar los analizadores humanos y complementar sus deficiencias, pero no necesariamente construir un humanoide no distingible de los humanos mediante el test de Turing [Moor, 1987] o los experimentos conceptuales de Searle [Searle, 1985].

En nuestra opinión los objetivos iniciales de la IA fueron excesivos porque se ignoraron: (1) El carácter general y precientífico del término y (2) las enormes diferencias constitutivas entre el “conocer humano” y el conocimiento que los humanos hemos sido capaces de hacer residir en una máquina de cristal de silicio semiconductor. La ignorancia del primero de estos puntos nos ha llevado a perseguir un objetivo excesivo y mal definido. La ignorancia del segundo punto nos ha llevado a olvidar que el trabajo real está en el desarrollo de arquitecturas, lenguajes y herramientas lógico-matemáticas que superponen organizaciones a la electrónica digital, de forma tal que a un observador humano le parece que la máquina “*es*” inteligente.

Curiosamente, este intento de añadir espectacularidad y nomenclatura cognitiva excesiva a nuestros modelos, a nuestros desarrollos formales y a nuestros programas y robots ha contribuido a oscurecer, al menos parcialmente, los sólidos resultados alcanzados por la IA en particular en técnicas de representación e inferencia, en aprendizaje, robótica, “visión” artificial y sistemas basados en el conocimiento (SBCs). Se han realizado avances importantes en las técnicas de modelado conceptual y formal, en la estructuración del conocimiento necesario para resolver una tarea en términos del “papel” que juegan los distintos elementos (roles) y del plan estratégico de descomposición del procedimiento de solución (“métodos”). También se ha avanzado en las técnicas de representación formal (lógica, reglas, marcos, objetos, agentes, redes causales, etc...) y en el tratamiento de la incertidumbre (redes bayesianas, sistemas borrosos) y en la solución de problemas para los que disponemos de más datos que conocimiento (redes de neuronas artificiales). Hay avances importantes en la búsqueda de inspiración en la biología (computación de membranas) y en la Física (computación cuántica); se ha alcanzado la frontera de la nanotecnología y se investiga en biomateriales como soporte físico de un cálculo. Finalmente, cuando las soluciones propuestas por la IA son valiosas, enseguida las integra la informática convencional, y de esto hay ejemplos en dominios tan diversos y relevantes como la robótica industrial, la medicina, el arte, la educación o la WEB. Es decir, independientemente de los excesivos objetivos iniciales y de la carga cognitiva de su nomenclatura, los logros alcanzados por la IA durante los últimos cincuenta años, entendida como automatización de procesos con alto contenido cognitivo, son indiscutibles.

El resto de este capítulo está estructurado de la siguiente forma. Primero, distinguimos entre los objetivos de la IA como ciencia de los de la IA como Ingeniería del Conocimiento (IC). Después, comentamos de forma resumida la evolución histórica y conceptual haciendo énfasis en los aspectos metodológicos hasta llegar al estado actual caracterizado por la conveniencia de tres paradigmas básicos (simbólico, conexiónista y situado) y un cuarto de carácter sincrético o híbrido. A continuación, enumeramos algunas de las diferencias constitutivas básicas que justifican las discrepancias entre las expectativas iniciales de la gran hipótesis de la IA y los resultados alcanzados por la IC. Esta distinción entre la perspectiva científica de la IA y la aplicada nos da pie a una serie de recomendaciones que creemos que pueden ayudar al avance de ambas.

1.2 La IA como Ciencia y como IC

Aunque lo usual en IA e IC es mezclar conceptos cognitivos y del lenguaje natural (intención, propósito, ontología, semántica, emoción, memoria, aprendizaje,...) con otros computacionales (modelos, inferencias, roles, entidades abstractas y operadores lógico-matemáticos, tablas, autómatas, programas,...) suponiendo que estos conceptos tienen el mismo significado y las mismas funcionalidades en computación que en humanos, lo cierto es que no es así. Urge entonces ayudar al lector a distinguir entre los objetivos de la IA como ciencia e IA como ingeniería (IC) (véase la Figura 1.1) para que sepa de qué hablamos cuando usamos estos conceptos (cuál es su referente) y evitar así los equívocos asociados a suponer que la semántica y la causalidad del lenguaje natural y la neurofisiología son equivalentes a la semántica y la causalidad de un lenguaje formal y del hardware que lo soporta. Esta mezcla de entidades cognitivas y biológicas con otras formales y abstractas, junto con la asignación arbitraria de significados (el “no saber llevar bien la contabilidad”, nos dice Maturana [Maturana, 1975]) es una de las causas fundamentales de la disparidad entre el optimismo excesivo de los objetivos iniciales y los nada despreciables resultados actuales de la IA. Una visión más realista de los problemas y un lenguaje más ajustado a las matemáticas, la lógica y la ingeniería electrónica ayuda a plantear los problemas y a resolver aquellos para los que tenemos solución en el estado actual de conocimiento. Cargar de nomenclatura la parte no computable del conocer humano no cambia las cosas. Prácticamente, todo el lenguaje de la IA y la IC ha sido tomado de la biología en general y de la neurofisiología, la psicología cognitiva y la filosofía en particular, y esto no está mal siempre que no se olvide la diferencia de significados porque estimula el proceso de búsqueda de soluciones. Es nefasto, sin embargo, cuando se da por resuelto un problema por el sólo cambio de nomenclatura.

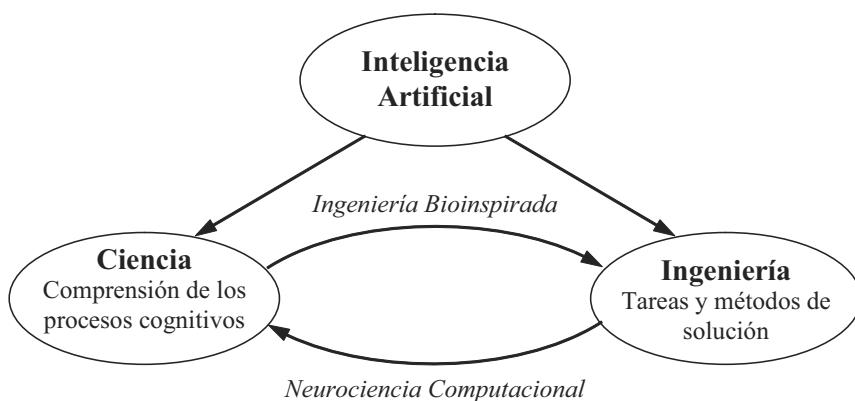


Figura 1.1: Distinción entre IA como ciencia e IA como ingeniería.

1.2.1 IA como Ciencia

Entendida como ciencia la tarea de IA es una tarea de análisis. Su fenomenología engloba el conjunto de hechos asociados a la neurología y la cognición, desde los niveles subcelular y neuronal a los mecanismos y organizaciones superpuestas de las que emergen las funciones globales de percepción, memoria, lenguaje, decisión, emoción y acción que han dado lugar a lo que llamamos *comportamiento inteligente en humanos*.

La perspectiva científica de la IA busca una teoría computable del conocimiento humano. Es decir, una teoría en la que sus modelos formales puedan ejecutarse en un sistema de cálculo y tener el mismo carácter predictivo que tienen, por ejemplo, las ecuaciones de Maxwell en el electromagnetismo. No es sorprendente entonces que con estos objetivos consideremos excesiva la conjectura fuerte de la IA. Dotar a la biología y a la psicología del carácter de ciencia experimental que tiene la física es un objetivo deseable pero de difícil consecución. Entre otras razones porque todavía no disponemos de los datos necesarios ni, posiblemente, de las matemáticas adecuadas. La labor que creemos que debe de realizar en esta tarea la computación en general y la IA en particular es dotar de herramientas conceptuales y formales a la Neurofisiología y la Ciencia Cognitiva. Es decir, potenciar la “Neurociencia Computacional” y todos los procedimientos experimentales de estimulación, clasificación, e interpretación y predicción de resultados.

Siempre ha habido un fuerte debate asociado a las relaciones entre la IA y la cognición con las analogías clásicas mente-programa, cerebro-hardware, y las consiguientes posturas a favor y en contra de la equivalencia entre pensar y computar. Aquí no vamos a entrar en estos temas, porque sólo nos interesa distinguir entre los objetivos científicos y los aplicados de la IA para contribuir a aclarar dónde están los problemas reales en el segundo caso, en la IC. Queda fuera del alcance de este tema los debates psicológicos y filosóficos sobre las relaciones mente-cerebro, e incluso las grandes preguntas sobre si algún día podrán o no pensar las máquinas. De momento, vamos a centrarnos en resolver cada vez problemas más complejos en el sentido de Leibniz. Vamos a ver cómo abordamos la falta de claridad en las especificaciones funcionales, la imprecisión, la incertidumbre y el aprendizaje.

1.2.2 IA como Ingeniería

La rama aplicada de la IA, conocida como Ingeniería del Conocimiento, tiene unos objetivos más claros y alcanzables a corto y medio plazo. Sin embargo, tiene también grandes dificultades comparada con las otras ingenierías de la materia y la energía por dos razones fundamentales. La primera es que, como alternativa a la materia y a la energía, el nuevo objeto formal de la IC es el *conocimiento* y éste, como la *información*, es pura forma. Sólo usa la energía como soporte, pero el mensaje está en la estructura relacional y en el consenso entre los distintos observadores externos que deberán de dotar del mismo significado a los símbolos formales y físicos que constituyen un cálculo.

El conocimiento es ahora objeto de observación, modelado, formalización y transformación por procesos de su mismo nivel o de metaniveles superiores (aprendizaje).

Además, si queremos que ese conocimiento sea reutilizable, debe de ser impersonal, transferible, verificable experimentalmente y con la misma capacidad de predicción de una ley física. Desafortunadamente, no estamos seguros de disponer de las matemáticas necesarias para formalizar el conocimiento de los procesos cognitivos de la misma forma que la física dispone del cálculo diferencial e integral.

La segunda razón de las dificultades de la IC en comparación con las dificultades usuales en otras ingenierías es que la IC no puede apoyarse en una sólida *teoría del conocimiento* porque todavía no disponemos de esa teoría. Así, estamos queriendo hacer aviones sin un sólido conocimiento de la física de fluidos. Por consiguiente, parece razonable dejar el tiempo necesario para que la parte teórica de la IA contribuya a obtener una teoría computable del conocer humano y, mientras tanto, redefinir los objetivos de la IC de forma más modesta, teniendo en cuenta el carácter limitado, incompleto y poco preciso del conocimiento del que disponemos sobre los dos tipos de tareas que aborda la IC: (1) Tareas básicas e inespecíficas usuales en humanos, independientemente de su actividad profesional, tales como ver, oír, interpretar el medio, planificar, aprender, controlar las acciones encaminadas a moverse y manipular un medio, etc. y (2) Tareas científico-técnicas en dominios estrechos (diagnosticar en medicina, configurar y diseñar sistemas, etc...). Es crucial aceptar inicialmente las limitaciones en el alcance, las funcionalidades y la autonomía de estos sistemas de IA. Estas limitaciones están asociadas al desconocimiento de la neurofisiología y la lógica de la cognición y a las diferencias constitutivas entre el cuerpo biológico y el robot, entre el lenguaje natural y el lenguaje formal, entre la semántica y la sintaxis.

En la mayoría de los desarrollos de la IC llamados Sistemas Basados en el Conocimiento (SBCs) se procede de acuerdo con los siguientes pasos (véase la Figura 1.2): (1) Se parte de una descripción en lenguaje natural de las interacciones de un humano con el entorno en el que se desarrolla la tarea que queremos sintetizar. Es decir, se parte del método usado por el experto humano para “resolver” esa tarea. (2) Después se modela esta descripción usando diferentes metamodelos a los que llamamos paradigmas (simbólico o representacional, conexiónista o situado). Cada una de estas formas de modelado conceptual es esencialmente un procedimiento de descomposición de la tarea en subtareas hasta llegar al nivel de inferencias primitivas que son aquellas componentes del razonamiento que ya no necesitan una descomposición posterior (“seleccionar”, “comparar”, ...) porque ya se pueden implementar usando sólo conocimiento del dominio. (3) El tercer paso es la reescritura formal de las inferencias y los “roles” estáticos y dinámicos de acuerdo con el paradigma elegido. Esta elección es consecuencia del balance entre los datos y el conocimiento disponibles para resolver cada tarea concreta, del tipo al que pertenecen esos datos disponibles (etiquetados o no etiquetados) y del tipo de conocimiento que demanda la aplicación. Hay dos tipos esenciales de conocimiento (situado y no situado), función de la naturaleza de la interfaz necesaria para acoplar un sistema de IA con su medio. El conocimiento situado está asociado a aquellas situaciones en las que la interacción con el medio la lleva a cabo un sistema electromecánico (un robot), por lo que hay que tener en cuenta los sensores y los efectores que limitan el repertorio de posibles comportamientos “inteligentes”. Alternativamente, el conocimiento no situado está asociado a aquellas otras situaciones en las que la interfaz entre el sistema de IA y el medio es humana, por

lo que no hay que preocuparse de sensores ni de efectores. (4) Finalmente, la última etapa del desarrollo de un SBC es programar los operadores. Para aquellas tareas en las que el interfaz es físico y no humano (sensores y efectores de un robot concreto), es imprescindible implementar también el conocimiento asociado al “cuerpo” soporte del cálculo.

Esta visión de la IA como ingeniería ha demostrado su utilidad como procedimiento general de solución de problemas de diagnóstico, planificación y control en un gran número de dominios de aplicación. Bajo su paraguas se han desarrollado procedimientos de representación de conocimiento declarativo explícito y de su uso en inferencia, bibliotecas de métodos, ontologías y otros servidores de componentes de modelado y formalización reutilizables.

Otra caso diferente es la potencial validez de esta aproximación de ingeniería para explicar los procesos cognitivos. Ya hemos mencionado que no vamos a entrar aquí en ese debate. El lector interesado puede seguir la pista a autores tales como Clancey [Clancey, 1997, 1999], Dreyfus [Dreyfus, 1979, 1994], Searle [Searle, 1989] o Edelman [Edelman, 1987], entre otros.

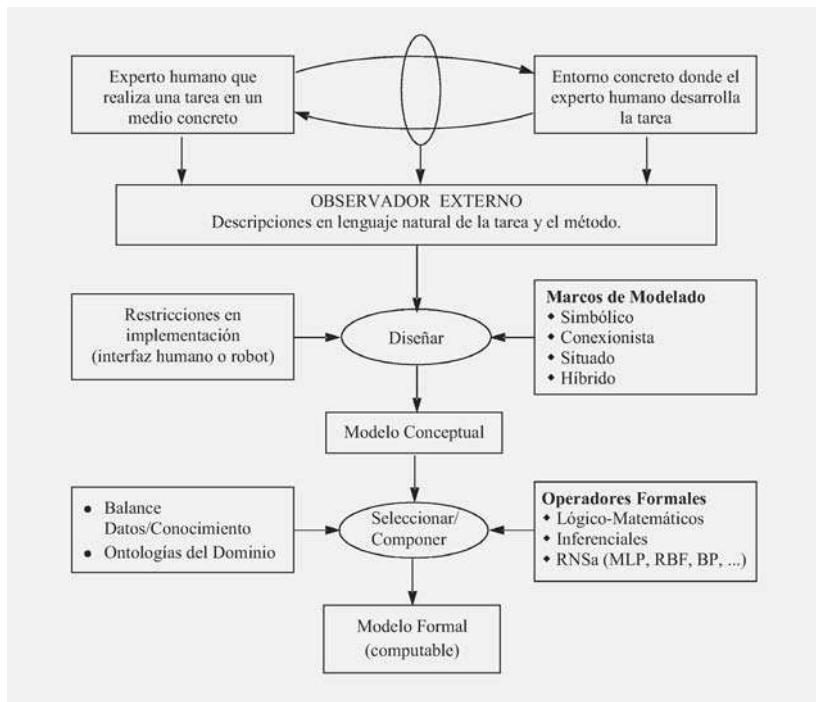


Figura 1.2: Esquema cualitativo de la IC. Se modela a partir de descripciones en lenguaje natural del procedimiento usado por un humano para resolver una tarea.

1.3 Perspectiva histórica: fundamentos y metodología

Es curioso observar ahora, cincuenta años después de acuñar el término IA, lo poco que se ha investigado en sus fundamentos y en metodología, y la excesiva prisa mostrada en desarrollar aplicaciones. Los paradigmas dominantes en la actualidad (simbólico, conexionista, situado e híbrido) tienen sus fundamentos en los trabajos previos a 1956 llevados a cabo por la Cibernetica, desde 1943. En ese año se publicaron los tres trabajos que pueden considerarse de carácter fundacional:

- *A Logical Calculus of the Ideas Immanent in Nervous Activity.* McCulloch, W.S., Pitts, W. [McCulloch y Pitts, 1943].
- *The Nature of Explanation.* Craik, K. [Craik, 1943][15].
- *Behavior, Purpose and Teleology.* Rosenblueth, A., Wiener, N., Bigelow, J. [Rosenblueth y otros, 1943; Wiener, 1947].

En el primero, W.S. McCulloch y W. Pitts introducen el concepto de neurona formal que ha dado origen al paradigma conexionista. En el segundo, K. Craik introduce los fundamentos del paradigma simbólico al interpretar el conocimiento humano en términos de descripciones declarativas y modulares de entidades simbólicas de alto nivel y de un conjunto de reglas inferenciales usadas para manipular esas descripciones simbólicas. Finalmente en el tercer trabajo, A. Rosenblueth, N. Wiener y J. Bigelow introducen las bases del paradigma situado al interpretar el comportamiento inteligente en términos de un conjunto de mecanismos de realimentación. Posteriormente, el trabajo de la escuela de W.S. McCulloch, incluyendo la relevante aportación de J. von Neumann, crea las bases de la visión cibernetica de la IA, que busca los fundamentos de la inteligencia en la enorme red de mecanismos genéticos, moleculares, neuronales, metabólicos y sociales de los que emerge. El trabajo de A. Turing en 1950 complementa los fundamentos del paradigma simbólico al proponer un procedimiento experimental de medir la inteligencia contenida en un “programa de IA” (lo que hoy conocemos como “test de Turing”).

Tras estos antecedentes ciberneticos, en 1956 se acuña el término IA y se abandona el conexionismo y el interés por el análisis del comportamiento y de los mecanismos soporte y se hace énfasis en el paradigma simbólico, que parte de descripciones de alto nivel para intentar dotar de inteligencia a un programa, que siempre se supone que va a tener un usuario humano, por lo que no hay que preocuparse del cuerpo del “agente” soporte del cálculo, ni de los “detalles” del sistema conexionista que se encarga de ejecutar ese programa.

El optimismo inicial de la IA estuvo basado en limitar su dominio de interés a los micromundos formales (“mundo de los bloques”) y a los sistemas capaces de ser descritos de forma completa con métodos lógicos, mediante procedimientos generales de búsqueda heurística, análisis medios-fines y “solucionadores generales de problemas”. Pronto se reconoce que el mundo real es más complejo y se empieza a hacer énfasis

en las técnicas de representación del conocimiento y de su uso posterior en inferencia, junto con el desarrollo de lenguajes de programación orientados a cada una de las técnicas de representación (lógica - prolog, reglas - lisp, marcos - orientación a objetos).

En la década de los setenta se inicia la explosión de aplicaciones de la IA en términos de sistemas basados en reglas a los que se les llama primero “Sistemas Expertos” y después “Sistemas Basados en Conocimiento”. El reconocimiento de la insuficiencia de la lógica como herramienta única de representación da lugar al desarrollo de otras formas de representación e inferencia mediante redes causales y asociativas (semánticas, neuronales y bayesianas) y marcos, objetos y agentes.

Aunque en la etapa dominada por el paradigma simbólico (1956-1986) hay un número relevante de trabajos basados en los principios de la cibernetica y el conexionismo [Craik, 1943; McCulloch y Pitts, 1943], tales como la teoría modular de autómatas probabilísticos, las memorias asociativas, el reconocimiento de caracteres, los sistemas autoorganizativos y la traducción automática, es en 1986 [Rumelhart y otros, 1986] cuando renace con fuerza la aproximación neuronal al problema de la inteligencia. En la década de los 90 [Arkin, 1998; Brooks, 1991] aparece también, en el contexto de la robótica, el interés por la aproximación situada, cerrando así el lazo histórico que comenzó en 1943, con el reconocimiento de la necesidad de usar los tres paradigmas tanto en la perspectiva teórica como en la aplicada. La dimensión del problema así lo exige.

Tal como comentábamos al comienzo de este apartado es curioso que durante todos estos años hayan sido mínimos los esfuerzos en el fortalecimiento de los fundamentos de la IA, más allá de los desarrollos en torno al proyecto SOAR (1969-1991) [Rosenbloom y otros, 1993] y de algunas críticas de filósofos y psicólogos interesados en la IA, como Dreyfus, Clancey o Searle. De hecho, tal como ya hemos mencionado, el debate más importante en IA no ha estado relacionado con la Ingeniería, sino con la potencial validez del paradigma simbólico para explicar los procesos cognitivos.

Algo análogo ha pasado con los aspectos metodológicos en los que empezó afirmando que el desarrollo de los SBCs era un “arte”, que no habían reglas ni procedimientos normalizados e invariantes, reproducibles y reutilizables. En nuestra opinión, sólo hay tres movimientos dignos de interés en el aspecto metodológico, más allá de las propuestas iniciales de Leibniz, Turing y McCulloch (véase la Figura 1.3): (1) El énfasis en la reutilización de componentes de modelado, a través del desarrollo de bibliotecas de tareas y métodos, (2) el desarrollo de ontologías y servidores de terminología unificada y (3) el trabajo de Newell, Marr, Maturana y Varela para especificar un marco de niveles y dominios de descripción de un cálculo que permite especificar de forma inequívoca qué parte del conocimiento termina residiendo en la arquitectura de un computador y cuál permanece fuera, en el dominio del observador externo y, por consiguiente todavía no es computable. La Figura 1.4 muestra este esquema de niveles y dominios de descripción de un cálculo y la forma usual en la que nos movemos dentro del marco, desde el dominio del observador del nivel de conocimiento hasta el dominio propio del nivel de los símbolos.

Sobre el nivel físico (el hardware del computador y del robot) se superpone el nivel de los símbolos (el programa) y sobre este el nivel de conocimiento (en nomenclatura

de Newell) o el nivel de la “teoría del cálculo” (en nomenclatura de Marr). Cada nivel es autónomo y se tiene que poder reescribir en términos de las entidades y relaciones del nivel inferior, dejando fuera parte de su semántica. Para explicar esta distinción entre la semántica y causalidad propias de cada nivel y las que no son computables. J. Mira y A.E Delgado [Mira y Delgado, 1987] introdujeron en 1987 la distinción en cada nivel entre el *dominio propio del nivel* donde la semántica es interna y las cosas ocurren “como tienen que ocurrir” (los contadores cuentan, los multiplexos multiplexan, ...), y el *dominio del observador externo* (nosotros), donde la semántica es arbitraria y tenemos libertad para asociar significados a símbolos.

Un aspecto importante para entender la parte computable de la inteligencia humana es detallar el conocimiento que se queda fuera cuando vamos atravesando fronteras de dominios y niveles en el proceso de reducción de un modelo conceptual, primero a un modelo formal y después a un programa.

Nuestro consejo es que a la hora de valorar las funcionalidades reales de un programa o un robot con supuesta inteligencia tengamos clara la distinción entre lo que pertenece al dominio propio (es decir lo que reside en la CPU del computador y en el cuerpo del robot) y lo que pertenece al dominio del observador (es decir las etiquetas lingüísticas y la semántica). Así, se nos hará evidente la distinción entre el nivel de inteligencia que realmente hemos sido capaces de computar y aquellos otros componentes de la inteligencia humana que sólo existen en el lenguaje natural y en la mente del usuario e intérprete del programa.

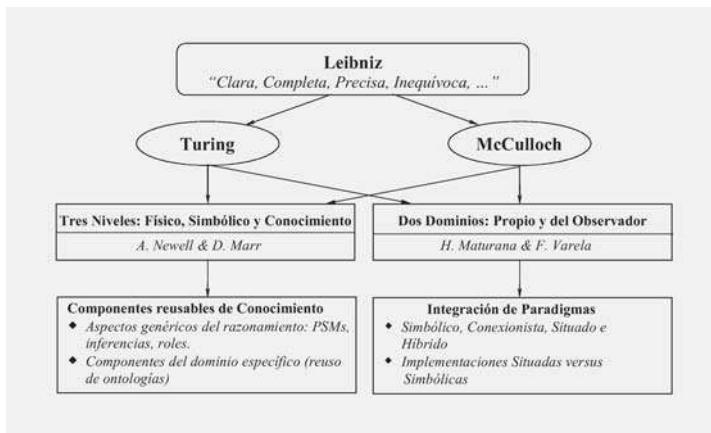


Figura 1.3: Pasos metodológicos en el desarrollo de la IA.

1.4 Paradigmas actuales en IA

Entendemos el concepto de paradigma en el sentido de Kuhn [Kuhn, 1971], como una aproximación metodológica a la IA y a la IC que ha sido consensuada entre un amplio grupo de profesionales del campo que la consideran como la forma normal de

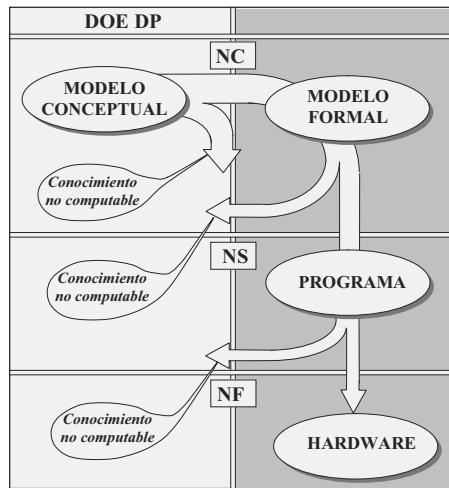


Figura 1.4: Niveles y dominios de descripción de un cálculo.

hacer ciencia o ingeniería. Este consenso en un paradigma concreto supone que se está de acuerdo sobre cuáles son los objetivos y la forma de alcanzarlos. Es decir, sobre cómo plantear las cuestiones y los experimentos, sobre los métodos para resolver los problemas y sobre los mecanismos de explicación y las hipótesis que nos van a dejar satisfechos. Paradigma es sinónimo de forma de abordar la solución de un problema.

Centrándonos en la IC, hemos aceptado al comienzo del capítulo que sus tareas básicas eran modelar conocimiento, formalizar los modelos, programar los operadores formales e implementar físicamente el soporte de esos programas (el “cuerpo” del robot). Por consiguiente, cada paradigma en IC es en esencia una forma de modelar, formalizar, programar e implementar conocimiento, junto con la hipótesis de partida acerca de qué entendemos por conocimiento.

Aunque veremos al final de este apartado que sólo hay dos tipos de paradigmas, los basados en *representaciones* y los basados en *mecanismos*, es usual distinguir cuatro paradigmas básicos:

1. Simbólico o representacional.
2. Situado o reactivo.
3. Conexionista.
4. Híbrido.

Describimos ahora de forma resumida estos paradigmas mencionando las situaciones en las que son más adecuados en función del balance entre datos y conocimientos disponibles.

1.4.1 El paradigma simbólico

Todo paradigma empieza tomando postura sobre qué entendemos por conocimiento. En el caso del paradigma *simbólico*, llamado también representacional, se considera que todo el conocimiento necesario para resolver una tarea de diagnóstico, planificación, control o aprendizaje, por ejemplo, puede representarse usando descripciones declarativas y explícitas en lenguaje natural formadas por un conjunto de “conceptos”, los hechos, y otro conjunto de reglas de inferencia que describen las relaciones estáticas y dinámicas conocidas entre esos hechos. Así, *razonar* en el paradigma simbólico es equivalente a especificar un conjunto de *reglas de manipulación* de los conceptos de entrada al SBC que genera el resultado del razonamiento, la inferencia.

El conocimiento se separa del experto humano que supuestamente lo posee, se acepta que la descripción en lenguaje natural es suficiente y esa descripción *se descompone* en términos de entidades (sustantivos) y verbos inferenciales (seleccionar, abstraer, comparar, ...) y de los condicionales de control (si ... entonces ...). Finalmente, estas componentes se clasifican y usan de acuerdo con el papel que juegan en un conjunto de modelos estructurales de las tareas que pretendemos resolver.

Este paradigma ha sido el dominante desde 1956 hasta 1986, y en su evolución se observa una tendencia progresiva a separar los aspectos genéricos (tareas, métodos, inferencias y roles) de los conocimientos específicos de cada dominio de aplicación (medicina, robótica, educación, Web,...). La meta es aproximar la IC a las otras ingenierías, en particular a la ingeniería electrónica, donde nadie discute la utilidad de disponer de buenos almacenes de componentes reutilizables (contadores, ALUs, memorias,...) y de procedimientos sistemáticos de síntesis a partir de estos componentes, de un “modelo estructural de la tarea” (esquemas de conectividad en circuitos patrón) y de un conjunto de reglas que especifican el diseño a partir de un conjunto de especificaciones funcionales. Es usual distinguir tres tipos de tareas, *de análisis* (monitorizar, clasificar, diagnosticar, ...), *de síntesis* (planificar, configurar, diseñar, refinar, ...) y *de modificación* (reparar, controlar, supervisar, aprender, ...). Cada tarea se corresponde con un patrón muy general de razonamiento que especifica el tipo de problema, sus objetivos y las actividades necesarias para alcanzarlos. Por ejemplo, se entiende como tarea de análisis toda aquella para la que disponemos a priori de todas las soluciones posibles (las clases) y el trabajo está en, dado un conjunto de observables, especificar a cuál de esas clases pertenece o con cuáles de ellas y en qué grado son compatibles esos observables. Así, las tareas de análisis son esencialmente tareas de clasificación. Alternativamente, en las tareas de síntesis no disponemos “a priori” de los resultados posibles porque se trata de un proceso de diseño y construcción con restricciones. Finalmente, las tareas de modificación tienen que ver, en general, con el ajuste de parámetros en la estructura de otras tareas.

La componente tarea es demasiado amplia como para disponer de un “almacén de tareas resueltas”, por lo que a cada tarea se asocia un conjunto de *métodos* (“establece y refina”, “propón-critica-modifica”, “abstactae-compara-refina”, ...) que la descomponen en *subtareas* y especifican el control. El proceso de *descomposición* termina cuando se alcanza el nivel de *inferencias* primitivas, llamadas así porque se corresponden con procesos elementales de razonamiento que no necesitan más descomposiciones ya que

se pueden operacionalizar usando sólo el conocimiento específico de cada dominio de aplicación. Las entidades del dominio representan roles de entrada y/o salida en las distintas inferencias, tal como se ilustra en la Figura 1.5. Las inferencias cumplen un papel análogo al de los circuitos integrados en electrónica, en los que ya no tenemos que preocuparnos de su estructura interna sino sólo de su descripción funcional, externa.

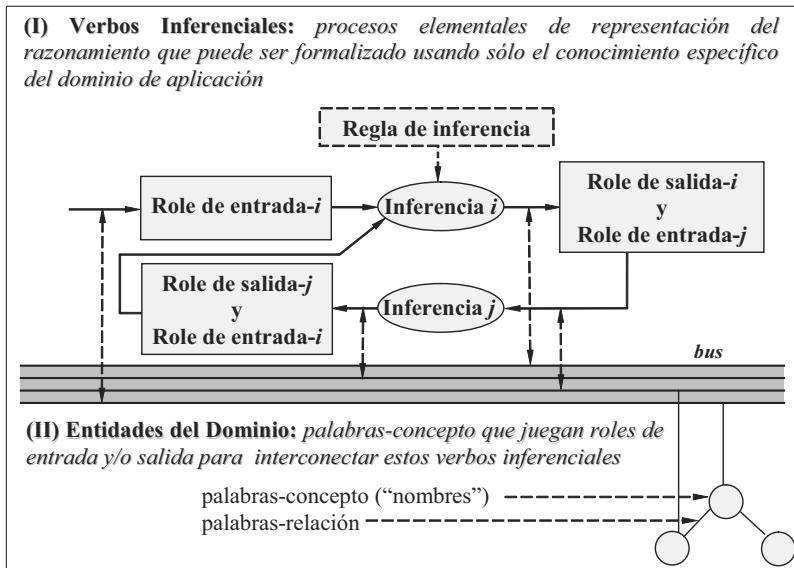


Figura 1.5: Esquemas inferenciales del paradigma simbólico, tras descomponer una tarea por un método concreto.

Obsérvese que subyacente a esta aproximación funcional al paradigma simbólico está la idea de circuito electrónico. La tarea (“contar”) se descompone por el método (síncrono) hasta llegar al nivel de inferencias (biestables) de forma que el esquema inferencial del razonamiento es el circuito del contador síncrono que se obtiene con un determinado esquema de conectividad entre varios biestables.

Obsérvese también que el concepto de role se aclara con la analogía electrónica. Una señal cumple el rol de reloj, por ejemplo, porque entra en un terminal tal que la estructura interna del circuito le obliga a satisfacer esa función. Lo mismo pasa con el rol de “observable” o “hipótesis diagnóstica” en una determinada inferencia (“abstraer”). Es decir, la estructura interna de las reglas que operacionalizan una inferencia definen el rol que van a cumplir las entidades de un dominio de aplicación que actúen como entradas o salidas de esa inferencia.

Para aquellos lectores que estén interesados en los antecedentes históricos del paradigma simbólico les recordamos que la primera descripción conocida aparece en la obra de Kenneth Craik que mencionamos al comienzo del apartado anterior (“La Naturaleza de la Explicación”). Craik distinguía tres procesos básicos en el razonamiento humano [Craik, 1943]:

1. *Traslación* de las entidades y relaciones del medio externo a una *representación* interna en términos de palabras, números u otros símbolos, como por ejemplo imágenes.
2. *Derivación* de nuevos símbolos a partir de los anteriores mediante procesos de inferencia inductiva, deductiva y, esencialmente, abductiva.
3. *Retraslación* de los nuevos símbolos derivados (inferidos) en términos de acciones que vuelven al medio externo. Alternativamente, la acción se sustituye por el reconocimiento a nivel reflexivo de la correspondencia entre estos nuevos símbolos inferidos y los eventos del medio externo a los que corresponden, dándonos cuenta de que se ha cumplido una predicción, evaluada en ese modelo dinámico del medio. Es decir, usamos nuestro modelo del medio (nuestra “base de conocimientos”) para predecir y evaluar distintas alternativas, pasamos a acciones las más ventajosa y actualizamos nuestro modelo del medio mediante aprendizaje. La Figura 1.6 muestra una versión actualizada de la propuesta de Craik, en la que no es difícil detectar analogías con las arquitecturas usuales en los SBCs.

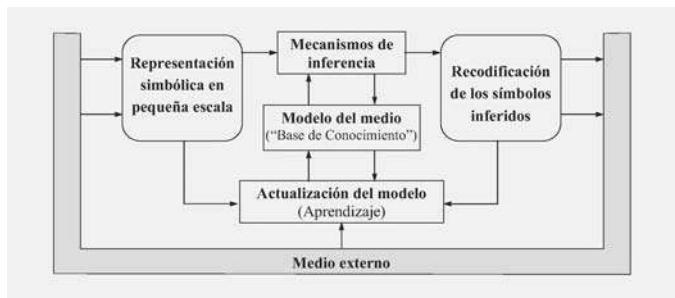


Figura 1.6: Versión actualizada de la arquitectura simbólica basada en la propuesta inicial de Craik.

Además de la vertiente funcional que hemos descrito, el paradigma simbólico es usado también por las orientaciones o objetos y a sistemas multiagente, dentro de la IA distribuida. En todos los casos, la aproximación simbólica es adecuada para todas aquellas aplicaciones en las que disponemos de conocimiento suficiente para especificar las reglas inferenciales y en aquellos procesos de aprendizaje inductivo en los que también disponemos de conocimiento para especificar las meta-reglas.

1.4.2 El paradigma situado

El paradigma situado, llamado también reactivo o “basado en conductas”, enfatiza el hecho de que toda percepción y toda acción están estructuralmente acopladas, a través de sensores y efectores concretos, a un medio externo e interno también concretos. Así, las componentes de modelado del conocimiento no son ahora conceptos cognitivos de alto nivel semántico (“hipótesis”, “observables”, “diagnósticos”, …), sino

elementos más sencillos claramente distinguibles por un observador externo que ve cómo un agente físico (un robot, por ejemplo) interactúa con su medio, tal como ocurre en la etología. Observamos que un robot “está quieto”, “se mueve”, “gira a la derecha”, “evita un obstáculo”, “acerca un manipulador a un objeto”, “lo coge”, “lo transporta”, “lo suelta”, etc. Éstas son las conductas elementales que queremos mimetizar.

Esta aproximación ascendente al problema de la IC tiene sus raíces en la cibernetica (recordemos que el título de la obra de W.S. McCulloch es “*Embodiments of Mind*” [McCulloch, 1965]) y ha sido promovida en la IA en la década de los 90 por investigadores procedentes del campo de la robótica [Arkin, 1998; Brooks, 1991; Murphy, 2002], la psicología [Clancey, 1997, 1999] y la biología [Varela, 1979, 2002]. Los nombres de Brooks, Clancey y Varela son representativos de esta aproximación. El esquema de la Figura 1.7 resume la arquitectura reactiva más elemental. El sistema que queremos modelar se encuentra en un medio con el que cierra un lazo de realimentación mediante un conjunto de sensores y efectores específicos. Así, todo lo que no puedan representar esos sensores no existe para el sistema, en sentido estricto. Análogamente, el sistema no podrá realizar ninguna acción que no pueda ejecutarse a través de sus efectores. El resto de sus potenciales decisiones no existen a efectos de modificar el medio.

El siguiente paso en la especificación de la arquitectura consiste en aceptar que las señales de los sensores son procesadas espacio-temporalmente para detectar la presencia en el medio de alguno de los *esquemas* de un conjunto predefinido. Todos estos esquemas están especificados a un nivel muy bajo, próximo al lenguaje máquina y la electrónica para que su tiempo de cálculo sea muy pequeño y permita al agente operar en tiempo real, de forma *reactiva*, sin necesidad de gastar mucho tiempo “de-liberando”, como ocurría en el paradigma simbólico. A estos esquemas de entrada se les suele llamar “*percepciones*”.

De forma especular, cuando el agente decide la acción o secuencia de acciones que debe de ejecutar, basta con que las seleccione y active, porque estos patrones espaciotemporales de acción también están precalculados y permiten una conexión directa, condicionada o secuencial con las percepciones.

Finalmente, la *función de decisión* del agente (la inferencia y su control) es esencialmente un esquema de asociación (una tabla) o un autómata finito que, ante cada configuración de percepciones activadas y cada estado interno dispara la configuración de acciones correspondientes. Es decir, el paradigma reactivo sustituye la representación de la parte de condición de las reglas inferenciales por esquemas precalculados y sustituye el proceso inferencial en reglas y marcos por su autómata de control. La parte de acción de las reglas también está aquí precalculada.

Esta visión externa y funcional de la conducta de un agente físico no siempre está comprometida con los mecanismos subyacentes de los que emerge, con lo que termina siendo tan descriptiva como la simbólica. La Tabla 1.1 muestra la equivalencia entre las nomenclaturas en ambos paradigmas. Hay una segunda vertiente del paradigma situado, que está más cerca de la fisiología, la electrónica y las aplicaciones de tiempo real, y que se caracteriza por especificar a nivel físico los mecanismos concretos usados para sintetizar los detectores de los esquemas perceptuales, los generadores de

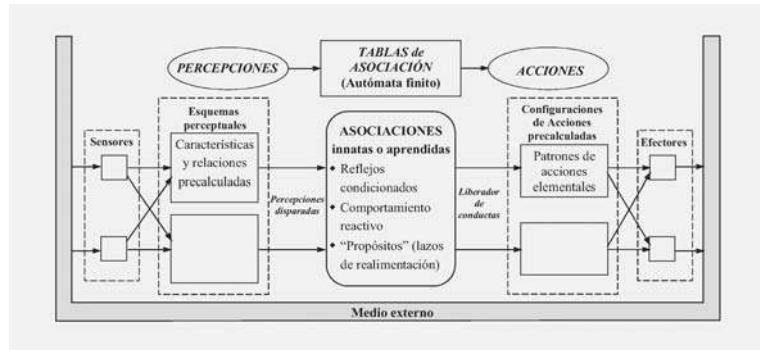


Figura 1.7: Arquitectura de la interacción de un agente inteligente con su medio, de acuerdo con el paradigma situado.

Paradigma simbólico	Paradigma situado
Tareas	Conjunto estructurado de conductas
Subtareas e inferencias	Conductas
Métodos	Tablas y autómatas de asociación percepción-acción
Esquemas inferenciales	Composición de conductas con mecanismos de solución de conflictos
Roles dinámicos de entrada	Liberadores de conducta
Roles dinámicos de salida	Patrones de salida
Control del esquema inferencial	Autómatas finitos

Tabla 1.1: Tabla de comparación entre la nomenclatura usada en el paradigma simbólico y la del situado.

los esquemas motores y el autómata asociado. La lógica del agente depende de las coordinaciones espacio-temporales entre los estados de actividad de estos dos grupos de mecanismos: Los perceptuales y los motores. Veremos más tarde que estas dos opciones (representaciones y mecanismos) también se dan en el paradigma conexiónista. Este paradigma situado se usa esencialmente en robótica y en aplicaciones de tiempo real. Cuando aumenta la complejidad del sistema se hace evidente la necesidad de soluciones híbridas, con componentes reactivas (rápidas) y componentes deliberativas (más lentas).

1.4.3 El paradigma conexiónista

En el paradigma conexiónista (las llamadas redes de neuronas artificiales, RNAs), el problema de la representación del conocimiento se realiza mediante el uso de líneas numéricas etiquetadas para la entrada y salida de la red y el problema de la inferencia se resuelve mediante un clasificador numérico de naturaleza paramétrica en el que el valor de esos parámetros se ajusta mediante un algoritmo de aprendizaje supervisado o no supervisado [Mira y Delgado, 1995a, 2002].

La arquitectura de un agente conexionista es modular, organizada en capas, con gran número de procesadores elementales (“neuronas”) fuertemente interconectados, que evalúan una sencilla función de cálculo local, en general la suma ponderada seguida de una sigmoide o cualquier otra función de decisión no lineal que acota el rango dinámico de la respuesta de cada unidad.

Las características distintivas de esta forma de modelar conocimiento, operacionalizarlo y programarlo son las siguientes:

1. Todos los problemas resueltos con RNAs tienen las características de un clasificador numérico adaptivo que asocia los valores de un conjunto de *observables* (representados mediante líneas numéricas etiquetadas) con los valores de otro conjunto más reducido de *clases*, representadas también por las salidas de las neuronas de la última capa que, a su vez, son también líneas numéricas etiquetadas.
2. Una parte importante del conocimiento disponible se corresponde con la fase de *análisis de los datos*, en la que somos nosotros quienes decidimos cuáles van a ser las variables de entrada y salida, qué tipo de cálculo local es el más adecuado, cuál es el número de capas y unidades por capa más adecuado, cómo se deben inicializar los pesos, etc.
3. Es importante conocer el balance entre datos y conocimiento disponible y la naturaleza de esos datos, que pueden ser etiquetados o no etiquetados. Los datos etiquetados (de los que se conoce la respuesta de la red) se usan en aprendizaje supervisado y en las fases finales de validar y evaluar la red. Los datos no etiquetados se usan para un preproceso y para el aprendizaje autoorganizativo, en donde la red trata de hacer explícito todo el conocimiento subyacente a la regularidad estadística de esos datos.

La Figura 1.8 muestra la arquitectura básica del modelado conexionista que corresponde a un grafo paralelo y dirigido en el que los nodos están ocupados por las neuronas y los arcos por los pesos ajustables. Tras la obtención de las líneas numéricas de entrada, la primera capa construye una base funcional, $\{\phi_k(\bar{x})\}$, y la segunda capa construye las funciones de salida, $y_j(\bar{x})$, como combinación lineal o no lineal de la representación de la entrada, \bar{x} , en términos de esa base funcional,

$$y_j(\bar{x}) = \sum_{k=1}^n w_{jk}(t) \cdot \phi_k(\bar{x})$$

Finalmente, las soluciones numéricas de la RNA, las salidas de las unidades de la última capa, se interpretan en términos de las etiquetas asociadas a las clases.

Es importante no olvidar el carácter numérico estricto del paradigma conexionista. La red sólo asocia números en términos de la naturaleza de las funciones usadas en la primera y en la segunda capa. Estas funciones pueden ser *lineales* (Fourier, autovalores, análisis de componentes principales) o *no lineales* (funciones de base radial, polinomios o desarrollos de Wiener-Volterra).

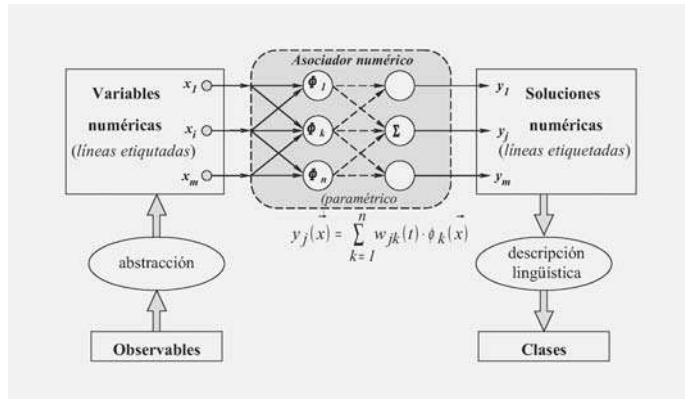


Figura 1.8: Arquitectura general de las RNAs como aproximadores paramétricos ajustables entre dos representaciones numéricas.

Al igual que en el paradigma situado, también en el conexionismo hay dos vertientes. La primera, y más usual en IC, es la que hemos descrito. Aquí la arquitectura es fija y la red actúa como un clasificador de propósito general. Hay otra vertiente, basada en mecanismos que está más cerca de la biología, porque tiene en cuenta de forma muy directa el conocimiento sobre la anatomía y fisiología de cada circuito neuronal concreto. En este sentido hablamos de conexionismo bioinspirado. Podemos mencionar varios ejemplos clásicos: las redes de inhibición lateral, los arcos reflejos, los circuitos de habituación y sensitización y los generadores de patrones de respuesta, donde la conectividad específica, los lazos de realimentación y el carácter excitador o inhibidor de cada contacto sináptico son determinantes en la función que calcula la red. Esta aproximación ascendente a la inteligencia suele ser más modesta que la simbólica y nos habla de mecanismos de adaptación de un agente a su medio, lo que equivale a aceptar que la inteligencia puede considerarse como una forma superior de adaptación construida sobre otras formas de adaptación más elementales y compartidas con el resto de la escala filogenética.

1.4.4 El paradigma híbrido

No es fácil encontrar un problema de IA para el que dispongamos de forma clara, precisa, completa e inequívoca de todo el conocimiento necesario para su solución. De hecho, en aquellos casos en los que así fuera ya no se trataría de un problema de IA, sino de computación convencional. Tampoco es frecuente que no sepamos nada para empezar a elaborar las especificaciones funcionales. La realidad es que la mayor parte de los problemas son de naturaleza *híbrida*, por lo que su solución también deberá ser híbrida, usando los datos y el conocimiento disponibles junto con los métodos y técnicas más adecuados para rentabilizar esos recursos. Es decir, en el control de un robot podremos necesitar aproximaciones reactivas y declarativas, y técnicas simbólicas, borrosas y neuronales. Esta es la idea de *sincretismo* que caracteriza a las

aproximaciones híbridas en IC. El cuerpo de conocimientos en IC ofrece una *suite* de métodos y técnicas de modelado, formalización, programación e implementación física. La naturaleza del problema y el balance entre datos y conocimientos son los criterios que guían en cada caso la combinación específica de medios más adecuada para su solución.

Las primeras propuestas [Hillario y otros, 1995] sobre la integración de los paradigmas simbólicos y conexionista proponían arquitecturas como las de la Figura 1.9, en las que las redes neuronales actuaban como *preprocesadores*, *postprocesadores* o *coprocesadores* subordinados a un procesador simbólico central. En otras ocasiones, se han usado las RNAs para obtener reglas simbólicas a partir de un conjunto de datos [Kuncicky y otros, 1992; Towell, 1992; Towell y Shavlik, 1994]. En estos casos se usa el conocimiento disponible para especificar la arquitectura de la red que posteriormente se entrena y finalmente, se trasladan a formato de reglas la matriz de pesos de la red entrenada. El antecedente de las reglas se corresponde con la expresión lógica o analítica (suma ponderada) y el consecuente suele ser de naturaleza binaria, entendiendo que se satisface una regla cuando se dispara la neurona que la representa.

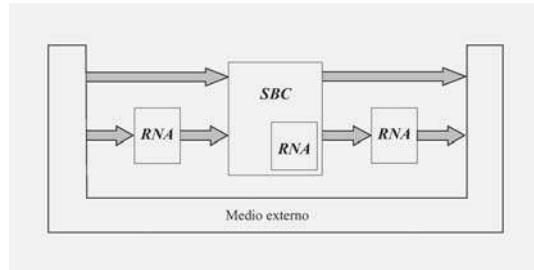


Figura 1.9: Ejemplo de arquitectura híbrida simbólica-conexionista en el que la RNA actúa como preprocesador, coprocesador o postprocesador de un sistema basado en reglas.

Otra forma usual de arquitectura híbrida es la de los *sistemas neuroborrosos* que engloban formas de modelar conocimiento y operacionalizar inferencias en las que se mezclan componentes neuronales y borrosas. Así, pueden usarse técnicas borrosas para “adquirir” el conocimiento, obteniendo una representación en términos de conjuntos borrosos y etiquetas lingüísticas de las distintas variables de entrada y salida del sistema. También pueden usarse entradas numéricas (parámetros que definen una función de pertenencia, valores discretos resultado de muestrear el universo objeto) y tratarlas después con operadores neuronales convencionales (sumas y productos). Finalmente, también pueden definirse esquemas de cálculo neuroborrosos en los que se mantiene el modelo estructural de neurona y las topologías de interconexión y se sustituyen los operadores analíticos usuales en la computación neuronal por operadores borrosos (min, max). Subyacente a los componentes borrosos del modelo existe también, en prácticamente todos los casos, un conocimiento simbólico usado para especificar las variables y sus rangos, para definir las funciones de pertenencia y para diseñar la topología inicial de la red neuronal.

Una parte importante de los problemas con los que se ha encontrado la IC al intentar integrar distintos paradigmas tienen que ver con la proximidad de la decisión a la fase de implementación. De hecho, en muchos proyectos se termina integrando sobre una plataforma que encapsula los distintos tipos de módulos (independientemente de la naturaleza simbólica, neuronal, situada o borrosa de los mismos) y garantiza el interfaz en los módulos encapsulados por ejemplo, usando arquitectura multiagente. Si, por el contrario, abordamos la tarea de integración a nivel de conocimiento y en el dominio del observador externo, como con cualquier otra tarea, el problema queda más claro y las soluciones más inequívocas, porque muchas de las aparentes diferencias entre paradigmas se difuminan al seguir los pasos usuales de modelado y formalización.

El criterio general siempre es el balance entre datos y conocimiento y la decisión entre métodos, técnicas y operadores representativos de los distintos paradigmas se debe hacer en aquella fase de descomposición de la tarea en la que se tenga claro ese balance para cada dominio de aplicación concreto. A la espera de desarrollos metodológicos más precisos, podemos guiarnos por los siguientes criterios generales:

1. Primero debemos analizar las exigencias computacionales del problema que queremos resolver y el conjunto de recursos de los que disponemos para su solución. Hay situaciones en las que en esta fase inicial ya podemos decidir qué PSM es más adecuado para resolver la tarea (clasificación jerárquica, perceptrón multicapa, clasificación heurística, red autoorganizativa, etc...), tal como se ilustra en la parte superior de la Figura 1.10.
2. Sin embargo, la situación más usual es que necesitemos seguir descomponiendo la tarea hasta el nivel de subtareas o inferencias primitivas antes de tener claro el balance entre datos y conocimiento y, por consiguiente, la forma más adecuada de operacionalizar esas inferencias. Por ejemplo, hemos podido empezar a descomponer una tarea de clasificación mediante el método de “clasificación jerárquica” dando lugar a dos inferencias básicas “establece” y “refina”. Es ahora cuando puede quedar claro que disponemos de conocimiento suficiente para “establecer” (por lo que operacionalizamos esta inferencia usando reglas simbólicas) y en cambio no disponemos de conocimiento suficiente para “refinar”, por lo que usamos un método neuronal, probabilístico o borroso para operacionalizar esta segunda inferencia.
3. El siguiente paso es la operacionalización efectiva del esquema inferencial, con módulos simbólicos y neuronales, resultado de las decisiones de la fase anterior. En ambos casos, ya sólo usamos operadores lógico-relacionales o analíticos directamente reescribibles en términos de las entidades y relaciones de un lenguaje de programación. Para ello tenemos que hacer compatibles las estructuras de datos y los roles (“observable”, “clase”, “medida”, “distancia”, “diagnóstico”, “hipótesis”, “sospecha”, ...) que participan en el intercambio de información entre módulos simbólicos, conexiónistas, situados y borrosos.

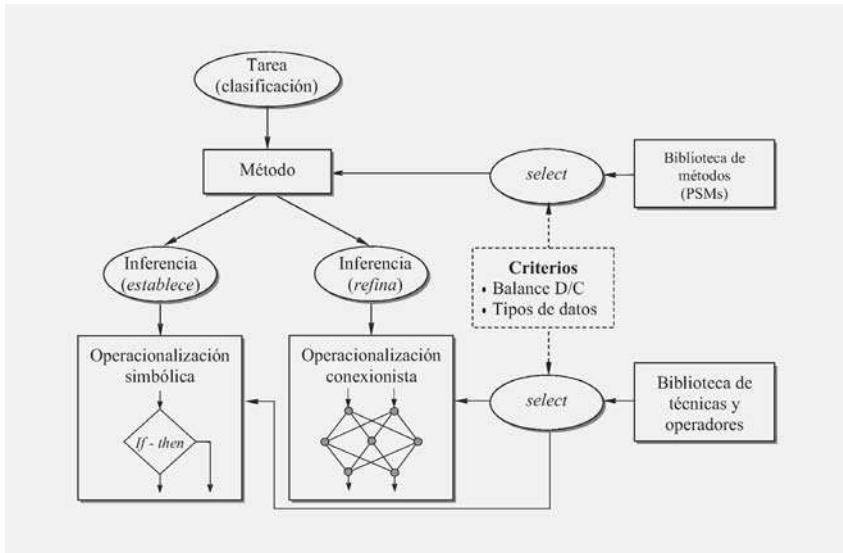


Figura 1.10: Ilustración de un ejemplo de aproximación híbrida (neurosimbólica) en la fase de operacionalización de las inferencias del método de clasificación “establece y refina”.

1.5 El conocer humano y el conocer de las máquinas

Comenzábamos este capítulo de aproximación conceptual a la IA y la IC recordando la disparidad existente entre los excesivos objetivos iniciales de la IA (“sintetizar inteligencia general en máquinas”) y los resultados obtenidos por la IC tras medio siglo de trabajo científico y técnico. También hemos analizado la principal causa de esa disparidad que no es otra que las enormes diferencias constitutivas entre los seres vivos y las máquinas. Esta primera reflexión nos ha llevado a separar los objetivos y los métodos propios de la IA considerada como ciencia, de los objetivos y métodos de la IC. El resto del capítulo se ha dedicado a resumir la evolución histórica de la IA y la IC hasta llegar a los cuatro paradigmas que caracterizan el estado actual del conocimiento en cuanto a aproximaciones metodológicas se refiere. Ahora es tiempo de volver a reflexionar sobre las diferencias entre el conocer humano y el conocer de las máquinas para dar fundamento a las recomendaciones del último apartado.

Todo conocer, nos dice Maturana, depende constitutivamente de la naturaleza del sistema que conoce, de sus entidades constituyentes y de sus relaciones con el medio al que está acoplado estructuralmente y con el que ha derivado evolutivamente. Así, hablar de inteligencia natural supone hablar de la vida, de los distintos niveles organizativos de la realidad biológica, social, histórica y evolutiva. La inteligencia está situada en un cuerpo y esa fisicalidad del sistema no puede eludirse. Por eso la IA sólo puede pretender cubrir la dimensión formal del pensamiento, la parte computable de la percepción, la acción y el razonamiento en humanos.

Es tan evidente que la naturaleza humana es distinta de la naturaleza del computador y del robot que casi parece innecesario el análisis comparativo. Sin embargo, dadas las consecuencias de esta analogía (cerebro-hardware, mente-software, pensar-calcular) vamos a mencionar algunas de las diferencias constitutivas entre el conocer de los sistemas vivos y la computación en máquinas con sensores y efectores electromecánicos usando el marco de niveles (físico, símbolos y conocimiento) y dominios de descripción de un cálculo (dominio propio de cada nivel y dominio del observador externo), que hemos mencionado en el apartado tercero de este capítulo [Mira, 2003, 2005a]. En cada nivel y en cada dominio el conocimiento acomodado en el mismo depende de la fenomenología que sus elementos constituyentes generan al operar. En el nivel físico de las máquinas las entidades constituyentes son circuitos lógicos y retardos que sólo permiten establecer distinciones binarias (0,1) sobre expresiones lógicas y transiciones de estado en autómatas finitos. Todo el resto del conocimiento que puede acomodar este nivel está asociado a la arquitectura (al lenguaje máquina). El cuerpo del computador es de cristal semiconductor, con arquitectura fija, estática y con semántica impuesta. Por el contrario, las entidades constituyentes del tejido nervioso (proteínas, canales iónicos, neuronas) permiten acomodar todo el conocimiento aportado por la genética, la evolución, la historia y la cultura. El soporte neurofisiológico del comportamiento inteligente es autónomo, autopoyético, dinámico y adaptivo. Su semántica es emergente y su arquitectura siempre está inacabada y, por consiguiente, finalmente es única e irrepetible para cada ser vivo y en cada instante de su existencia. Nos construimos al vivir.

También en el nivel de los símbolos hay diferencias esenciales. En los computadores programables convencionales usamos símbolos “fríos”, descriptivos, estáticos y sintáticos, con una gramática rígida. El significado de estos símbolos es arbitrario y no interviene en el cálculo, porque permanece en el dominio del observador externo. Su descripción más completa es la propuesta por Newell y Simon en su “Physical Symbol System Hypothesis” [Newell y Simon, 1976]. Por el contrario, el símbolo neurofisiológico es dinámico y está asociado a los mecanismos que lo generan e interpretan [Mira, 2005b; Mira y Delgado, 2006]. Sin el conocimiento de la historia evolutiva de un animal concreto (la rana) y de su medio (la charca) será difícil entender la representación simbólica que construyen sus células ganglionares (las “bug detectors”), por ejemplo. Finalmente, las diferencias en el nivel de conocimiento son las de observación más directa, tanto por introspección como por consideración del cuerpo de conocimientos acumulados por la genética, la fisiología, la psicología, la lingüística, la sociología y la filosofía. La inteligencia humana está asociada a la capacidad de adaptación al medio, al uso del lenguaje natural y al comportamiento basado en propósitos, intenciones y emociones. Si queremos usar aquí la metáfora computacional para afirmar que pensar es calcular, tenemos primero que formular los conceptos de cálculo intencional y cálculo semántico. Por el contrario, ya hemos recordado que al computador sólo pasa el “modelo formal subyacente” asociado a un lenguaje de programación para el que hemos sido capaces de desarrollar un traductor al lenguaje máquina. Es decir, todo cálculo es *sintáctico o lógico* y está descrito en extenso. Hoy por hoy no tenemos una idea razonablemente clara, consensuada y físicamente realizable de cómo podríamos construir un cálculo intencional y semántico, guiado por propósitos y significados.

Es cierto que todas las máquinas incorporan información y conocimiento grabado

en su estructura; justo el conocimiento que usó su diseñador de acuerdo con un plan y para satisfacer un propósito (la función de la máquina). También el computador incorpora conocimiento para realizar un conjunto amplio de tareas pero ese conocimiento está en su arquitectura, en los lenguajes de programación, en el conjunto de “reglas fijas” a las que llamamos algoritmos y en el conjunto de reglas variables llamadas “de aprendizaje” porque modifican el valor de los parámetros de las reglas fijas. Sin embargo, en todos estos casos, la máquina sólo ejecuta las reglas. Todo el resto del conocimiento permanece en el dominio del observador externo.

Una segunda fuente de disparidad entre el conocer humano y el conocer de las máquinas procede de nuestro desconocimiento de la fisiología y la lógica de la cognición. Nos faltan datos y teorías sobre el conocer humano. En cambio, “el conocer de las máquinas” lo conocemos perfectamente porque las hemos diseñado nosotros.

Si se hubiera distinguido claramente desde el principio la vertiente aplicada de la IA (resolver problemas en los que hay incertidumbre, conocimiento incompleto y necesidad de aprendizaje) de la vertiente teórica (comprender y formular la inteligencia humana), nos habríamos ahorrado muchos problemas. En particular, los ocasionados por la prisa excesiva en hacer ingeniería (desarrollar aplicaciones) sin disponer del soporte científico previo y necesario: una teoría bien fundada experimentalmente, invariante y con capacidad predictiva sobre el conocer humano. Esto ha llevado a una cierta superficialidad en las propuestas de la IC que han enmascarado la falta de teoría con un uso inadecuado del lenguaje. Muchas de las propuestas de la IA y la IC sólo existen en el lenguaje del observador. El uso no justificado de términos cognitivos, dando por supuesto que tienen el mismo significado en humanos que en computación ha contribuido enormemente a crear el espejismo de que el nombre de las etiquetas que usa el programador se corresponde exactamente con lo que el computador hace con la entidad abstracta subyacente a esa etiqueta. De hecho, es al revés, basta con intentar hacer ingeniería inversa de un programa concreto para comprobar que hay muchas descripciones compatibles con el mismo. También hay muchos significados compatibles con los símbolos usados en la formulación de un fenómeno físico usando una ecuación diferencial (campo eléctrico, magnético, gravitatorio, número de elementos de una población, etc.) y no por eso adscribimos las semánticas propias del electromagnetismo, la gravitación, la mecánica cuántica o la sociología poblacional a la semántica propia de las entidades abstractas de un texto sobre ecuaciones diferenciales (variables, conjuntos, sumas, restas, derivadas, integrales, ...), tal como se estudia en una Facultad de Matemáticas. La Facultad de Física es “otro edificio” que está al lado. Las Facultades de Sociología y Psicología también están en otros edificios próximos, pero no son el mismo que el de la Lógica y las Matemáticas.

Finalmente, una tercera fuente de discrepancia entre el conocer humano y el de las máquinas procede de la falta de herramientas formales y nuevos modelos de computación adecuados para modelar los procesos cognitivos, de forma análoga a cómo el cálculo diferencial permitió la formulación matemática de la Física.

En todos los desarrollos de IA e IC, al intentar reescribir formalmente los modelos conceptuales, sólo disponemos de las matemáticas heredadas de la física (el álgebra y el cálculo), la lógica y la teoría de autómatas, junto con algunos elementos de cálculo de probabilidades y estadística. La duda es si estas herramientas formales son

o no suficientes para describir los procesos cognitivos. En nuestra opinión, hacen falta nuevas herramientas formales para captar el carácter dinámico, adaptivo, impreciso, en ocasiones contradictorio, intencional y semántico del conocimiento humano. Algo análogo creemos que pasa también con los fundamentos de la computación (máquinas de Turing, y redes de neuronas formales en el sentido de McCulloch-Pitts) y la arquitectura von Neumann, que es posible que no sean suficientes (quizás ni siquiera los más adecuados) para modelar los procesos mentales. Quizás sea posible explicar el pensamiento sin computación. De acuerdo con Bronowski, si queremos modelar los procesos soporte de la inteligencia en términos computables, tendremos que reducirlos a un lenguaje formal y no está claro que la naturaleza de lo vivo permita esa reducción. Los nuevos desarrollos en el campo de la computación bioinspirada tienen en muchos casos esta misma limitación porque parten de ricos conceptos biológicos sobre membranas y ADN, por ejemplo, pero terminan en las tierras del álgebra y los autómatas. No hay entonces nuevos modelos de computación en sentido estricto. No hemos sabido salir del extenso y la sintaxis. No hemos sabido hacer computable la semántica.

1.6 Algunas sugerencias

Para contribuir a disminuir la disparidad entre objetivos y resultados en IA e IC y hacerlas ambas más robustas, presentamos de forma resumida algunas sugerencias. La primera sugerencia es establecer una *distinción* clara entre el concepto de inteligencia general en humanos y los objetivos realizables a corto y medio plazo por la IC. Es decir, separar los objetivos y la nomenclatura de la IA entendida como *ciencia* de los de la IA entendida como *ingeniería*. Ya hemos comentado en el apartado 1.2 nuestra visión de la IA como ciencia, que se basa en considerar que el término inteligencia es muy general y de carácter precientífico. Consecuentemente, nos parece conveniente: (1) descomponerlo en un conjunto de habilidades parcialmente autónomas y accesibles al estudio experimental (percepción, abstracción, razonamiento recursivo, auto-referencia, ...), (2) aumentar el esfuerzo dedicado al estudio de los fundamentos de cada una de esas habilidades y a la construcción de un soporte teórico adecuado para las mismas, (3) intentar desarrollar nuevas herramientas conceptuales, formales y computacionales para describir adecuadamente los procesos mentales que hayamos sido capaces de identificar, analizar y controlar experimentalmente.

En la IA entendida como ingeniería (IC), los objetivos y los procedimientos son muy diferentes. Aquí no buscamos la comprensión de la inteligencia humana, sino la posibilidad de reescribir en forma computable los procedimientos usados por los humanos para resolver un conjunto de problemas científico-técnicos, en general descritos de forma poco clara, imprecisa e incompleta. Para su solución, la IC usa todos los métodos y técnicas disponibles, igual que el resto de la informática, incluyendo los cuatro paradigmas actuales, y las combina en función del balance entre datos y conocimientos disponibles para cada aplicación específica.

Sin embargo, ahora que hemos establecido la distinción entre IA como ciencia e IC queremos hacer énfasis en la utilidad de las aproximaciones y, en particular, en mirar a la naturaleza y a las ciencias de lo vivo con un doble objetivo [Mira, 2005b, 2006; Mira y Delgado, 2006]. Por una parte, como fuente de inspiración para encontrar nuevos materiales como potencial soporte del cálculo (DNA, membranas, moléculas) y para formular nuevos mecanismos, nuevas estrategias de programación y nuevos modelos de potencial utilidad en IC. Esta orientación se conoce con el nombre de *computación e ingeniería bioinspiradas*. Por otro lado, es cada vez más necesario el trabajo interdisciplinario en el que la física, las ingenierías, la lógica, las matemáticas y, resumiéndolas a todas, la *computación*, ayuden a las ciencias de lo vivo a experimentar y formular teorías explicativas que cierren el lazo de realimentación proporcionando a la IA y la IC el fundamento científico que tanto necesitan. Esta orientación complementaria se conoce con el nombre de *Neurociencia Computacional* y, en un contexto más amplio, como *visión computacional de las ciencias de lo vivo*. En términos históricos estamos hablando de *Biocibernética* y *Biónica*.

1.7 Resumen

En este capítulo inicial se ha explorado el concepto de Inteligencia Artificial (IA) considerada como ciencia y como ingeniería del Conocimiento (IC). Tras un breve resumen de la evolución histórica del campo se ha revisado los cuatro paradigmas actuales (simbólico, conexionista, situado e híbrido) a través de los cuales se ha hecho una aproximación tanto al intento de comprender la inteligencia humana, como a la solución de una serie de problemas característicos del campo. Después, se ha reflexionado sobre las diferencias constitutivas entre el conocer humano y el conocer de las máquinas. Finalmente, se han hecho algunas recomendaciones, que creemos que pueden contribuir a aclarar y alcanzar los objetivos específicos de la IA y la IC, a la vez que se eliminan falsas expectativas resultado en su mayoría del uso de una nomenclatura cognitiva excesiva.

Por lo tanto, el propósito global del capítulo ha consistido en dar una visión razonablemente amplia de la IA y la IC para que pueda servir como contexto y marco de referencia en el que el lector pueda situar los contenidos de la mayoría del resto de los capítulos. Sin embargo, la posición que se toma en este capítulo sobre las hipótesis fuerte y débil de la IA, la conveniencia de separar sus objetivos de los de la IC y el consejo final de potenciar la interacción entre neurociencia y computación, son de la única responsabilidad del autor.

Referencias

- ARKIN, R.C.: *Behavior-based Robotics*. The MIT Press, 1998.
- BROOKS, R.A.: «Intelligence without Reason». *Informe técnico Memo N°. 1293*, MIT A.I., 1991.
- CLANCEY, W.J.: *Situated Cognition. On human knowledge and computer representations*. Univ. Press, Cambridge, 1997.
- CLANCEY, W.J.: *Conceptual Coordination*. Lawrence Erlbaum Associates, Pub. Mahwah, New Jersey, 1999.
- CRAIK, K.: *The Nature of Explanation*. Cambridge University Press, Cambridge, 1943.
- DREYFUS, H.L.: *What Computers Can't do: The Limits of Artificial Intelligence*. Harper Colophon Books. Harper & Row, Publishers, N. York, 1979.
- DREYFUS, H.L.: *What Computers Still Can't do*. The MIT Press, Camb. Mass, 1994.
- EDELMAN, G.M.: *Neural Darwinism: The theory of neural group selection*. Basic Books. N.Y., 1987.
- GARDNER, H.: *Frames of Mind: The Theory of Multiple Intelligences*. Basic Books. N. York, 2004.
- HILLARIO, M.; LLAMEMENT, Y. y ALEXANDRE, F.: «Neurosymbolic integration: Unified versus hybrid approaches». En: *The European Symposium on Artificial Neural Networks*, Brussels, Belgium, 1995.
- KUHN, T.S.: *La Estructura de las Revoluciones Científicas*. Fondo de Cultura Económica. México, 1971.
- KUNCICKY, D.C.; HRUSKA, S.I. y LACHER, R.C.: «Hybrid systems: The equivalence of rule-based expert system and artificial neural network inference». *International Journal of Expert System*, 4(3), 1992, pp. 281–297.
- MATURANA, H.R.: «The Organization of the Living: A theory of the Living Organization». *Int. J. Man-Machine Studies* 7, 1975, pp. 313–332.
- MCCARTHY, J.; MINSKY, M.L.; RICHESTER, N. y SHANNON, C.E.: «A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence». *Informe técnico Int. report*, Hannover, New Hampshire, 1955.
- MCCULLOCH, W.S.: *Embodiments of Mind*. The MIT Press. Cambridge, Mass, 1965.
- MCCULLOCH, W.S. y PITTS, W.: «A Logical Calculus of the Ideas Immanent in Nervous Activity». *Bulletin of Mathematical Biophysics Vol. 5*, Chicago Univ. Press, 1943, pp. 115–133.

- MIRA, J.: «Del Conocer Humano al Conocer de las Máquinas», 2003. Lección Inaugural del Curso 2003-04. UNED.
- MIRA, J.: «On the Physical Formal and Semantic Frontiers between Human Knowing and Machine Knowing». En: *Computer Aided Systems Theory. LNCS 3643; Moreno-Díaz, R., Pichler, F., Quesada Arencibia, A. (Eds.)*, pp. 1-8. Springer-Verlag, Berlin, 2005a.
- MIRA, J.: «On the Use of the Computational Paradigm in Neurophysiology and Cognitive Science». En: *Mechanisms, Symbols, and Models Underlying Cognition. LNCS 3561; J. Mira and J.R. Álvarez (Eds.)*, pp. 1-15. Springer-Verlag, 2005b.
- MIRA, J.: «On Some of the Neural Mechanisms underlying Adaptive Behavior». En: *Intelligent Data Engineering and Automated Learning. LNCS 4224; Corchado, E. Yin, H. Botti, V. Fyfe, C. (Eds.)*, pp. 1-15. Springer-Verlag, Berlin, 2006.
- MIRA, J. y DELGADO, A.E.: «Some Comments on the Antropocentric Viewpoint in the Neurocybernetic Methodology». En: *Proc. of the Seventh International Congress of Cybernetics and Systems, Vol. 2. London*, pp. 891-895, 1987.
- MIRA, J. y DELGADO, A.E.: «Computación Neuronal». En: *Aspectos Básicos de la Inteligencia Artificial. Cap. 11; J. Mira, A.E. Delgado, J.G. Boticario y F.J. Díez*, pp. 485-575. Sanz y Torres, Madrid, 1995a.
- MIRA, J. y DELGADO, A.E.: «Perspectiva Histórica Conceptual». En: *Aspectos Básicos de la Inteligencia Artificial*, pp. 1-51. Sanz y Torres, Madrid, 1995b.
- MIRA, J. y DELGADO, A.E.: «Computación Neuronal: Una Perspectiva Dual». En: *Fronteras de la Computación; Barro, S., and Bugarín, A.J. (Eds.)*, pp. 265-312. Dintel y Díaz de Santos. Santiago de Compostela, 2002.
- MIRA, J. y DELGADO, A.E.: «On how the computational paradigm can help us to model and interpret the neural function». *Natural Computing. ISSN 1567-7818; Springer Netherlands*, 2006.
- MOOR, J.H.: «Turing Test». En: *Encyclopaedia of Artificial Intelligence Vol II; Shapiro, S.C. (Ed.)*, pp. 1126-1130. J. Wiley & Sons, New York, 1987.
- MURPHY, R.R.: *Introduction to AI robotics*. MIT Press Camb. Mass., 2002.
- NEWELL, A. y SIMON, H.A.: «Computer Science as Empirical Inquiry: Symbols and Search». En: *Communications of ACM, 19*, pp. 113-126, 1976.
- RICH, E.: «Artificial Intelligence». En: *Encyclopaedia of Artificial Intelligence Vol. I; Shapiro, S.C. (Ed.)*, pp. 9-16. Wiley & Sons, N. York, 1990.
- ROSENBLOOM, P.S.; LAIRD, J.E. y NEWELL, A. (EDS): *The SOAR papers. Vol. I and II*. The MIT Press, 1993.

- ROSENBLUETH, A.; WIENER, N. y BIGELOW, J.: «Behavior, Purpose and Teleology». *Philosophy of Science 10*, 1943.
- RUMELHART, D.E.; HINTON, G.E. y WILLIAMS, R.J.: «Learning internal representations by error propagation». En: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. Foundations; Rumelhart, D.E. and McClelland, J.L. (Eds.)*, pp. 318–362. MIT Press, Cambridge, MA, 1986.
- SEARLE, J.: «Minds, Brains, and Programs». En: *Mind Design. Philosophy, Psychology, Artificial Intelligence; Haugeland, J. (Ed.)*, pp. 282–306. The MIT Press, 1985.
- SEARLE, J.: «Minds and Brains without Programs». En: *Mindwaves; Blakemore, C. and Greenfield, S. (eds.)*, Basil Blackwell, 1989.
- TOWELL, G.G.: «Symbolic knowledge and neural networks: insertion refinement and extraction». *Informe técnico 1072*, Univ. of Wisconsin-Madison, Computer Science Dept., 1992.
- TOWELL, G.G. y SHAVLIK, J.W.: «Refining Symbolic Knowledge using Neural Networks». En: *Machine Learning. Vol. IV; Michalshki, R., Tecuci, G. (Eds.)*, pp. 405–429. Morgan-Kaufman Pub. C.A., 1994.
- VARELA, F.J.: *Principles of Biological Autonomy*. The North Holland Series in General Systems Research. North-Holland, New York, 1979.
- VARELA, F.J.: «The Re-Enchantment of the Concrete». En: *The Artificial Route to Artificial Intelligence; Steels, L., and Brooks, R. (Eds.)*, pp. 11–22. Lawrence Erlbaum Assoc. Pub. Hilldale, 2002.
- WIENER, N.: *Cybernetics*. The Technology Press. J. Wiley & Sons, New York, 1947.

Parte II

REPRESENTACIÓN DE CONOCIMIENTO E INFERENCIA

Capítulo 2

Lógica y representación del conocimiento

Guido Sciavicco
Universidad de Murcia

2.1 Introducción: ¿Por qué la Lógica?

El objetivo de la IA es la construcción de sistemas, tanto *hardware* como *software*, que sean capaces de replicar aspectos de lo que se suele considerar inteligencia. Evidentemente, este objetivo está relacionado con la definición de la propia palabra *inteligencia*, de la que existe, aproximadamente, una definición por cada persona. En el presente capítulo, utilizaremos la siguiente definición:

La IA es el conjunto de técnicas, métodos, herramientas y metodologías que nos ayudan a construir sistemas que se comportan de manera similar a un humano en la resolución de problemas concretos.

La creación de máquinas inteligentes (en el sentido anterior) es un deseo que ha existido en el hombre desde las primeras civilizaciones, aunque ha sido a partir del siglo XX, con la llegada de los computadores, cuando ha dejado de ser algo mitológico y se ha convertido en un asunto científico y, en parte, una realidad. En este capítulo dejaremos aparte las cuestiones psicológico-filosóficas sobre la naturaleza y el sentido último de la IA, cuestiones que, por otra parte, no carecen en absoluto de importancia. En este sentido, cabe recordar el debate sobre la naturaleza físico-matemática de la inteligencia y sobre la cuestión de cuándo una solución algorítmica a un problema concreto deja de ser una *simulación* de la inteligencia (o sea, un algoritmo que se comporta como un humano a la hora de resolver un problema y que, a partir de la misma entrada, devuelve siempre la misma salida) y es posible decir que, por lo que concierne al problema en cuestión, somos capaces de construir una máquina que *razona* eficazmente como un humano [Caballero y otros, 2006].

A pesar de las distintas interpretaciones del término IA, todas las técnicas y las metodologías que han sido desarrolladas y explotadas a lo largo de la historia de esta disciplina tienen en común un mismo problema: la *representación del conocimiento*. Para explicar con un ejemplo qué significa representar (formalmente) el conocimiento, imaginemos que nunca hemos estado antes en Madrid, y queremos encontrar el camino más corto para ir desde una estación de metro a otra. Lo primero que se nos ocurre es recurrir a un plano de las líneas de metro de Madrid, que no es nada más y nada menos que una representación de las líneas de metro reales. Algunos de los motivos por los que recurrimos a una representación, en lugar de a la realidad, son evidentes: en primer lugar nos resultaría imposible abarcar todas las líneas de metro de un vistazo, y en segundo lugar, sólo olvidándonos de todos los detalles superfluos a nuestro objetivo (como la presencia de escaleras móviles o de máquinas expendedoras de billetes) y centrándonos en las características importantes con el fin de encontrar un camino, podríamos utilizar (seguramente, sin darnos cuenta) un *algoritmo* que resolviera el problema. Asimismo, resultan claras al menos dos características fundamentales de una representación adecuada del conocimiento: la representación debe de ser *esencial* (no contiene información inútil) y *formal* (en su interpretación lógica el término *formal* indica que, en el mismo contexto, símbolos iguales tienen igual significado, o semántica).

En este capítulo, nos centraremos en la parte de la IA que concierne el *razonamiento automático*. Para nosotros, razonar significa obtener conclusiones (correctas) a partir de ciertas premisas (que consideramos correctas). Si el método (sea cual sea) que utilicemos nos devuelve siempre conclusiones correctas, entonces diremos que el método (de deducción) es *correcto*; si, además, el método es capaz de devolvernos todas las conclusiones correctas, entonces lo llamamos *completo*. La corrección y completitud de los métodos deductivos no son las únicas características en las que estamos interesados; también debemos tener en cuenta:

- El poder expresivo del lenguaje que utilicemos.
- Las propiedades computacionales del método.
- El grado de síntesis de las expresiones.

Al conjunto *lenguaje + semántica* que nos sirve para representar el conocimiento relacionado con la capacidad de llevar al cabo ciertos razonamientos lo llamamos *lógica*. Tal vez el ejemplo más antiguo del uso de la lógica en el sentido formal es el clásico *silogismo aristotélico*: *Los hombres son mortales; Sócrates es un hombre; entonces, Sócrates es mortal*. Si queremos enseñar a una máquina cómo sacar la conclusión *Sócrates es mortal* a partir de las premisas, no podemos pensar en enseñar a la máquina *quién es* (era) Sócrates y qué significa ser mortal. Lo que podemos hacer es *representar* tanto Sócrates como el conjunto de todas las personas con los *símbolos* adecuados, por ejemplo, utilizando *variables* para denotar personas, y un *predicado* (unario) para denotar la propiedad de ser mortal, y traducir el razonamiento en un *lenguaje formal*.

- Premisa: *(todos) los hombres son mortales* = $\forall x(Hombre(x) \rightarrow Mortal(x))$.
- Premisa: *Sócrates es un hombre* = $Hombre(Socrates)$.
- Conclusión: *Sócrates es mortal* = $Mortal(Socrates)$.

Dentro del significado de la frase *todos los hombres*, se incluye también un hombre en concreto que llamamos Sócrates. Entonces, en el predicado $Hombre(x)$ podemos *instanciar* la variable x con una constante, es decir, *Socrates*, y aplicar la regla que la premisa representa, pudiendo deducir $Mortal(Socrates)$. Está claro que lo que acabamos de hacer es una deducción, y que su carácter formal nos permite decir que es *automatizable*. Es más, puesto que ninguno de los pasos de deducción depende directamente, por ejemplo, del hecho *Sócrates era un filósofo de la antigua Grecia*, el mismo esquema de razonamiento se puede utilizar para expresar el hecho, *si todos los elementos que tienen la propiedad A también tienen la propiedad B, y el elemento llamado a tiene la propiedad A, entonces a tiene la propiedad B*. Este proceso de generalización demuestra que el tratamiento lógico-formal del razonamiento tiene muy buenas propiedades que resultarán, como veremos, extremadamente útiles a la hora de aplicarlas.

Acerca de los objetivos del presente capítulo. Este capítulo *no* es un manual de lógica. Por lo tanto, no habrá demostraciones de los teoremas que enunciaremos, y sólo en algunos casos pondremos las ideas básicas para tales demostraciones. Además, este capítulo *no* es un manual de computabilidad. Es importante decir que la teoría de la computabilidad es una disciplina fuertemente relacionada con la lógica clásica; sin embargo, desde un punto de vista más práctico, es posible aprender las nociones básicas del razonamiento automático sin entrar en los detalles de la computabilidad y de los problemas de decidibilidad/indecidibilidad. Para una visión más amplia y completa, sugerimos, entre otros, el [Boolos y otros, 2002].

Notación. A lo largo de este capítulo, utilizaremos una notación consistente y común a la mayoría de los trabajos en lógica que hay en la literatura. Las *variables* que denotan elementos del dominio se indicarán con las últimas letras del alfabeto latino (por ejemplo, $x, y, z\dots$), o con las mismas letras enriquecidas con índices ($x_1, y_3, z_i\dots$). Las variables *proposicionales* de orden-0 se denotarán también con letras del alfabeto latino como $p, q, r\dots$, y cuando queramos denotar predicados de órdenes superiores utilizaremos símbolos como $p(x), q(x, y), \dots$. Un símbolo de predicado puede también ser denotado con su nombre y un superíndice que indique su *aridad*, es decir, p^1, q^2, \dots . Los cuantificadores serán denotados de acuerdo con la literatura clásica, es decir, \forall significa *para todo* y \exists significa *existe*. Las *fórmulas* lógicas se expresarán con letras griegas minúsculas ($\phi, \varphi, \psi, \dots$), posiblemente enriquecidas con subíndices cuando sea necesario (ϕ_1, ψ_k, \dots). Los conjuntos de fórmulas serán denotados con letras griegas mayúsculas (Γ, Δ, \dots), y la relación de *satisfacibilidad* o *verdad* en un modelo se escribirá con el símbolo \models . Es importante destacar que el símbolo \models se sustituye por el símbolo \Vdash en el caso de las lógicas modales. Los modelos, tanto proposicionales y de primer orden clásicos, como modales o temporales serán denotados con letras mayúsculas del alfabeto latino (M, N, O, \dots). Para terminar, cada vez que queramos hacer una representación intensional de un conjunto de fórmulas

bien formadas, es decir, explicar cómo se pueden obtener todas y solamente todas las fórmulas bien formadas de un lenguaje dado, utilizaremos una *gramática abstracta*, como en el siguiente ejemplo:

$$\phi ::= A \mid B \mid \phi \circ \psi,$$

donde indicamos que las fórmulas bien formadas en el presente contexto son todas y solamente todas las que tienen la forma: $A, B, A \circ B, (A \circ A) \circ B \dots$

Este capítulo se estructurará de las siguiente forma: En la próxima sección presentaremos la lógica proposicional clásica, su sintaxis, su semántica y algunos de los más importantes métodos deductivos, y destacaremos sus límites de expresividad. En la sección 2.3 seguiremos la misma estructura para el lenguaje de primer orden, mientras que en la sección 2.4 nos centraremos en posibles extensiones de las lógicas clásicas. En la sección 2.5 consideraremos un ejemplo de extensión hablando de lógicas temporales. Finalmente, en la sección 2.6 se resuelven algunos ejercicios, y en la sección 2.7 se proponen problemas para resolver por el lector¹.

2.2 Lógica proposicional

En esta sección nos centraremos en el lenguaje lógico más sencillo, es decir, el lenguaje de la lógica proposicional o de orden 0, denotada también como LP. Después de presentar su sintaxis y su semántica, nos centraremos en el poder expresivo de LP, evidenciando sus límites, y concluiremos explicando algunos de los métodos deductivos más importantes, sus propiedades computacionales, y sus posibles implementaciones. Una posible referencia bibliográfica es [Schechter, 2005].

2.2.1 Sintaxis y semántica

El lenguaje de la *lógica proposicional clásica* (LP), está basado en un conjunto numerable (no necesariamente finito) de *proposiciones AP* (del inglés Atomic Propositions), y sus fórmulas bien formadas son todas y solamente todas las fórmulas que se pueden obtener a través de la siguiente gramática abstracta:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi, \tag{2.1}$$

siendo $p \in AP$. Sabemos que existen otras *conectivas* distintas a \neg y \wedge , y sabemos también que el conjunto $\{\neg, \wedge\}$ es un conjunto *completo* de conectivas; demostraremos informalmente este resultado más adelante en esta sección. Por ahora, podemos pensar que las fórmulas de LP cuentan con un conjunto extendido de conectivas, es decir, el conjunto $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, por lo que la gramática abstracta se convierte en la siguiente:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \tag{2.2}$$

¹Los problemas indicados con un asterisco se consideran de nivel ‘difícil’.

donde $p \in \mathcal{AP}$. En las secciones siguientes, aprenderemos por qué es conveniente utilizar, cuando sea posible, el lenguaje más restringido posible, en lugar de un lenguaje con muchos símbolos.

El primer problema que se presenta a la hora de utilizar una lógica (en este caso LP) para el razonamiento automático, es el problema de distinguir entre una fórmula bien formada y otra que no lo es. En la literatura, precisamente en los trabajos sobre traductores y compiladores (véase, por ejemplo, [Muchnick, 1997]), es posible encontrar técnicas y algoritmo eficientes para esta tarea; de hecho, el problema es muy conocido en el ámbito de la informática teórica, y consiste, en general, en averiguar si una *frase* (conjunto ordenado de símbolos) pertenece o no a las frases que pueden ser producidas por una gramática dada. Por ejemplo, la fórmula

$$\phi = (p \wedge q) \vee (p \rightarrow \neg\neg(q \vee r))$$

está bien formada con respecto a la gramática (1.2), pero no con respecto a la gramática (1.1), mientras la fórmula

$$\psi = \neg(p \wedge \vee(p \rightarrow \neg\neg(q \vee r)))$$

no está bien formada para ninguna de las dos.

Una *proposición* es una expresión en lenguaje natural que sólo puede ser *falsa* (*F*) o *verdadera* (*V*) (en la semántica del sentido común). Por ejemplo, *el suelo está mojado* es una proposición, mientras *¿cuánto tarda el autobús en llegar?* no lo es. Reconocer las proposiciones y las conectivas en una frase del lenguaje natural permite obtener una formulación correcta, y pasar sin pérdida de información a un lenguaje formal como el de LP. Por ejemplo:

El robot se encuentra con el brazo mecánico vacío. Si el robot está sujetando algo en el brazo mecánico, no puede levantar nada más desde el suelo. El objeto A se encuentra en el suelo. El objeto B se encuentra en el suelo. Entonces, si el robot levanta del suelo el objeto A no puede levantar también el objeto B.

Como se deduce del párrafo anterior, el sujeto *robot* no aparece como objeto del discurso, y por lo tanto no necesitaremos un símbolo proposicional para él. La primera proposición es *el robot se encuentra con el brazo mecánico vacío*, que cumple la condición de poder ser solamente verdadera o falsa. Entonces, utilizamos un símbolo, digamos

$$p$$

para indicar la condición de tener el brazo mecánico vacío. Si a lo largo del razonamiento nos encontramos el problema de indicar que el brazo mecánico *no* está vacío, podemos utilizar la fórmula

$$\neg p,$$

o, alternativamente, podríamos utilizar un símbolo proposicional nuevo, digamos q , y añadir al conjunto de premisas la fórmula

$$q \leftrightarrow \neg p,$$

donde, como veremos en esta sección, el símbolo \leftrightarrow (*bicondicional*) se puede leer *si y sólo si*. La segunda premisa es *si el robot está sujetando algo en el brazo mecánico (si el brazo no se encuentra vacío), no puede levantar nada más desde el suelo*, que se puede interpretar como *si el robot recoge algo desde el suelo, el brazo no está vacío*. Es importante, en este punto, resaltar que en el discurso sólo aparecen dos posibles objetos, es decir, A y B . Entonces, debemos formalizar la anterior premisa teniendo en cuenta los dos casos; digamos que el símbolo proposicional r_A indica que el robot recoge (o ha recogido) el objeto A desde el suelo, y similarmente con el símbolo r_B . La fórmula que estamos buscando será

$$\neg p \rightarrow \neg r_A \wedge \neg r_B,$$

es decir, es una fórmula condicional cuyo *antecedente* es *el brazo no está vacío*, y cuyo *consecuente* es *el robot recoge A o recoge B*. Dejamos al lector completar este ejemplo en el Problema 5.

¿Cómo se *evalúa* una fórmula ϕ de LP? Un *modelo* (o *mundo*) es una asignación de valores de verdad a todos los símbolos proposicionales que aparecen en la fórmula ϕ en el conjunto de fórmulas del que estamos hablando. Por ejemplo, en la fórmula $\phi = p \wedge (q \rightarrow r)$, sólo aparecen los símbolos p, q y r ; por lo tanto, la asignación $M = \{p = V, q = V, r = F\}$ es un posible modelo de ϕ . Naturalmente, dada una fórmula ϕ , no todas las asignaciones posibles de valores de verdad son tales que ϕ se evalúe como verdadera: decidir si éste es el caso o no depende de cómo se relacionan entre sí los símbolos proposicionales en la fórmula, es decir, depende de las conectivas. Si consideramos una fórmula ϕ y cualquier modelo M , podemos preguntarnos si M hace que la fórmula ϕ sea evaluada como V . En tal caso, se dice que M *satisface* ϕ y lo denotaremos con la expresión $M \models \phi$. Se dice que una fórmula es *satisfacible* si existe, al menos, un modelo que la satisface; en caso contrario, se dice que la fórmula es *insatisfacible*. Para saber si un modelo M satisface una fórmula ϕ aplicamos las siguientes reglas de forma recursiva:

- $M \models p$ si y sólo si $p = V$ en M ;
- $M \models \neg\psi$ si y sólo si no es verdad que $M \models \psi$;
- $M \models \psi \wedge \varphi$ si y sólo si $M \models \psi$ y $M \models \varphi$.

Por ejemplo, si evaluamos la fórmula $p \wedge \neg q$ en un modelo $M = \{p = V, q = V\}$, obtenemos que $M \models p \wedge \neg q$ si y sólo si $M \models p$ y $M \models \neg q$, es decir, si y sólo si $p = V$ en M (que es cierto) y no es verdad que $M \models q$ (que no es cierto). Por lo tanto, $M \not\models p \wedge \neg q$. Para poder estudiar el comportamiento global de una fórmula es necesario evaluarla en todos los modelos posibles. Una forma habitual de representar

todas las evaluaciones es mediante su *tabla de verdad*. La tabla de verdad de una fórmula (véase la Tabla 2.1) es una enumeración completa del valor de la fórmula para todos sus modelos. En general, la tabla de verdad tendrá 2^n filas, donde n es el número de variables proposicionales distintas que aparecen en la fórmula.

p	q	$\neg q$	$p \wedge \neg q$
V	V	F	F
V	F	V	V
F	V	F	F
F	F	V	F

Tabla 2.1: Tabla de verdad para la fórmula $p \wedge \neg q$.

Se dice que dos fórmulas ϕ y ψ son *equivalentes* si y sólo si tienen el mismo conjunto de modelos que las satisfacen, es decir, si para todo modelo M de ϕ y ψ se cumple que $M \models \phi$ si y sólo si $M \models \psi$. Esta propiedad es muy importante en el ámbito del razonamiento automático. Como hemos visto al principio de esta sección, el conjunto $\{\neg, \wedge\}$ es completo para LP. Esto quiere decir que cualquier fórmula de LP es equivalente a otra fórmula expresada únicamente con estas conectivas. Para empezar, es muy sencillo demostrar que el resto de conectivas que hemos visto anteriormente, es decir, \vee , \rightarrow y \leftrightarrow , pueden expresarse mediante las conectivas \wedge y \neg . De manera muy informal, podemos decir que $p \vee q$ es verdad si y sólo si no es cierto que tanto p como q son falsas al mismo tiempo, lo que significa que $p \vee q$ es lo mismo que $\neg(\neg p \wedge \neg q)$. De la misma manera, el condicional $p \rightarrow q$ se puede leer como *si p entonces q*, lo que significa que no podemos decir nada si no se da el caso que p es verdad, mientras que en ese caso (es decir, p es verdadero) sí estamos seguros de que también q es cierto; por otro lado, en lógica clásica, ante la ausencia de información, en caso de que p sea falsa, se considera que $p \rightarrow q$ es verdadera, y por lo tanto es equivalente a $\neg p \vee q$. En el Problema 4 el lector puede resolver el caso del bicondicional. Por otra parte, de forma general, dada una fórmula ϕ de LP con una tabla de verdad como la que se muestra a continuación:

p_1	p_2	...	ϕ
V	V	V	...
V	V	F	...
V	F	V	...
V	F	F	V
F	V	V	...
F	V	F	...
F	F	V	...
F	F	F	F
...

podemos obtener una fórmula equivalente expresada únicamente con conectivas \neg y \wedge utilizando el siguiente método: de dicha tabla, nos interesan las filas en las que ϕ

está evaluada a V ; si elegimos cada una de ellas, podemos obtener una fórmula parcial que se comporte como ϕ simplemente utilizando la conjunción de todas los símbolos proposicionales p_i evaluadas a V en esa fila, y todas las letras proposicionales $\neg p_j$ evaluadas a F en la misma fila. Uniéndolo con una disyunción (\vee) todas las fórmulas parciales, que ya sabemos cómo expresar a través de \neg y \wedge , obtenemos una fórmula perfectamente equivalente a ϕ .

Algunas fórmulas de LP cumplen unas condiciones especiales. Una *tautología* (o fórmula *válida*) es una fórmula de LP tal que, *independientemente* del modelo en la que la evaluamos, siempre es verdadera, como por ejemplo $p \vee \neg p$. La negación de una tautología, que por lo tanto siempre es falsa, recibe el nombre de *contradicción*. Las tautologías son el eje central de cualquier lógica; de hecho, el problema de la *decisión* para una lógica consiste en determinar si una fórmula dada (bien formada) es, o no, una tautología en esa lógica. Desde un punto de vista práctico, podemos considerar que un *razonamiento* es un conjunto de fórmulas $\Gamma = \{\phi_1, \dots, \phi_n\}$ del cual se puede deducir una *conclusión* ϕ , es decir,

$$\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi. \quad (2.3)$$

Supongamos ahora que queremos preguntarnos si el razonamiento (2.3) es *válido* o no. Por lo que hemos dicho anteriormente, cualquier fórmula es válida si y sólo si su negación es una contradicción, es decir, en nuestro caso, si y sólo si la fórmula

$$\phi_1 \wedge \dots \wedge \phi_n \wedge \neg \phi \quad (2.4)$$

es una contradicción o, si preferimos, no hay ningún modelo que la satisfaga. Razonar automáticamente significa exactamente comprobar la validez (bien directamente, o bien a través de la negación) de ciertas fórmulas. Los métodos deductivos, que veremos más adelante, se dividen en dos clases:

1. Métodos *sintácticos*: buscan una demostración formal de la validez de una fórmula, a través de reglas de deducción (aunque, como veremos, el método sintáctico llamado *resolución* es un método de refutación de fórmulas);
2. Métodos *semánticos*: buscan un contraejemplo para una fórmula, intentando demostrar que la fórmula no es satisfacible (en la mayoría de los casos, aunque existen excepciones como las tablas de verdad).

Los métodos de la segunda clase son los que normalmente se utilizan en las aplicaciones prácticas, ya que son fácilmente implementables y optimizables (la resolución es un caso aparte, siendo sin duda el método más utilizado en la práctica; otra característica importante desde el punto de vista práctico de los métodos semánticos es que se utilizan de forma muy extendida en el estudio de nuevas lógicas).

2.2.2 Poder expresivo y límites de la Lógica Proposicional

Como hemos visto, LP es un lenguaje que permite expresar ciertos razonamientos y cuyos métodos deductivos, como veremos, son particularmente sencillos y aptos

para la implementación. Además, el carácter *finito* de LP garantiza la decidibilidad del problema de la validez de fórmulas y conjuntos de fórmulas en LP. Pero ¿cuáles son los límites de LP? Consideremos el ejemplo del robot de la sección 2.2.1. Podemos identificar básicamente dos problemas.

1. Hemos podido expresar las premisas solamente con la condición de conocer con antelación el número (y los nombres) de los objetos que hay en el suelo. ¿Qué ocurriría si elimináramos esta hipótesis?
2. No hemos tenido en cuenta ninguna componente *temporal* o *espacial* en la descripción de las acciones. ¿Es posible hacerlo con algún lenguaje extendido?

2.2.3 Métodos deductivos semánticos y coste computacional

El método del árbol semántico. Uno de los métodos más importantes para decidir la validez de una fórmula en LP es el método del *árbol semántico*, también conocido como tablero semántico o simplemente *tableau* (una buena referencia bibliográfica para saber más sobre esta técnica es [D'Agostino y otros, 1999]). Como hemos dicho en la sección anterior, una posible técnica para demostrar que ϕ es una fórmula válida consiste en demostrar que $\neg\phi$ es una fórmula para la cual no hay modelos posibles que la satisfacen. Esto significa que, cada vez que intentamos construir un modelo para $\neg\phi$ asignando valores de verdad, nos encontramos con alguna contradicción. El ejemplo más sencillo consiste en demostrar que

$$p \wedge \neg p$$

no es satisfacible. Se observa que, si damos el valor $p = V$, por la semántica de la conectiva \neg , la fórmula $\neg p$ se evalúa a falso, y la conjunción de dos fórmulas de las cuales una es falsa, resulta falsa; se obtiene exactamente el mismo resultado si intentamos dar el valor $p = F$. El método del árbol semántico es un método *no determinista*: esto significa que se elige de forma casual entre varias posibilidades (o *ramas*), y se intenta completar la asignación de valores de verdad; si esto es posible, entonces hemos encontrado un modelo que la satisface, mientras que si se encuentra una contradicción, volvemos atrás e intentamos otra posibilidad. Si hemos terminado todas las posibilidades sin encontrar ninguna asignación completa, podemos concluir que la fórmula no es satisfacible. Las reglas dependen directamente de la semántica de las distintas conectivas:

- Conectivas *conjuntivas*, es decir, $\neg\neg$ y \wedge , expanden la rama actual: por ejemplo, si estamos evaluando $\phi \wedge \psi$, entonces, en la misma rama, evaluaremos ϕ y ψ .
- Conectivas *disyuntivas*, es decir, \vee , \rightarrow , \leftrightarrow , abren distintas ramas: por ejemplo, si estamos evaluando $\phi \vee \psi$, entonces, elejimos seguir o bien una rama en la que evaluamos ϕ , o bien una en que evaluamos ψ .

Cada vez que una fórmula (o un nodo) se expande, su estado cambia, pasando de *activo* a *no activo*. Una fórmula no activa no se considera nunca más para el desarrollo

del árbol semántico. La única excepción a esta regla (la que convierte una fórmula de activa a no activa) la veremos en la próxima sección. El número máximo de subfórmulas de una fórmula ϕ en el lenguaje LP es lineal ($O(n)$) en la longitud (número de símbolos) de ϕ . Por lo tanto, en el peor de los casos se emplea un tiempo polinomial en la longitud de ϕ en desarrollar una rama entera de un árbol semántico para ϕ . Esto significa que el algoritmo basado en árboles semánticos es un algoritmo polinomial no determinista, es decir, pertenece a la clase de complejidad NP . También se puede demostrar que el problema de la validez de LP pertenece a la clase de problemas NP -completos (lo que implica que no existe un algoritmo que pueda obtener un mejor orden de complejidad computacional), y que el método es correcto y completo. En el Algoritmo 2.1 se muestra un algoritmo para la deducción por medio de un árbol semántico. Terminaremos esta sección con un ejemplo; supongamos que queremos demostrar que $\phi = ((p \rightarrow q) \wedge (q \rightarrow \neg r) \wedge p) \rightarrow \neg r$ es válida. Como hemos visto, esto supone demostrar que la fórmula $\psi = \neg((p \rightarrow q) \wedge (q \rightarrow \neg r) \wedge p) \rightarrow \neg r$ no es satisfacible. Primero, observamos que se trata de un condicional; esto significa que ψ es satisfacible si y sólo si es satisfacible el conjunto $\{((p \rightarrow q) \wedge (q \rightarrow \neg r) \wedge p), \neg \neg r\}$ o, lo que es lo mismo, si y sólo si es satisfacible la fórmula $(p \rightarrow q) \wedge (q \rightarrow \neg r) \wedge p \wedge \neg \neg r$. En la Figura 2.1 podemos ver el desarrollo del árbol para esta fórmula, que termina con todas las ramas cerradas.

Algoritmo 2.1 Un algoritmo basado en árboles semánticos.

- 1: Entrada: $\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi$
 - 2: Transforma la entrada en la fórmula $\phi_1 \wedge \dots \wedge \phi_n \wedge \neg \phi$
 - 3: bucle
 - 4: si Existe una rama abierta y una regla que se puede aplicar a una fórmula ϕ en esa rama entonces
 - 5: Aplica la regla correspondiente a ϕ y convierte ϕ en NO ACTIVA
 - 6: fin si
 - 7: si Existe una rama que contiene una contradicción entonces
 - 8: Cierra la rama
 - 9: fin si
 - 10: fin bucle
 - 11: si Existe por lo menos una rama abierta entonces
 - 12: Devuelve SATISFACIBLE
 - 13: fin si
 - 14: si Todas las ramas están cerradas entonces
 - 15: Devuelve NO SATISFACIBLE
 - 16: fin si
-

El método de resolución. La idea básica del *método de resolución* para el lenguaje de LP es la siguiente. Supongamos que Γ_1 y Γ_2 son dos conjuntos de disyunciones de fórmulas de LP (p.e., $\Gamma_1 = \{\phi_1, \phi_2, \dots, \phi_k\} = (\phi_1 \vee \phi_2 \vee \dots \vee \phi_k)$, y ϕ una fórmula de LP. Si sabemos que las fórmulas $\Gamma \vee \phi$ y $\Gamma \vee \neg \phi$ son válidas a la vez, las cuales podemos expresar en notación de conjuntos como:

$$\Gamma_1 \cup \{\phi\} \quad \Gamma_2 \cup \{\neg \phi\},$$

entonces podemos deducir que también es válido (*regla de resolución*)

$$\Gamma_1 \cup \Gamma_2.$$

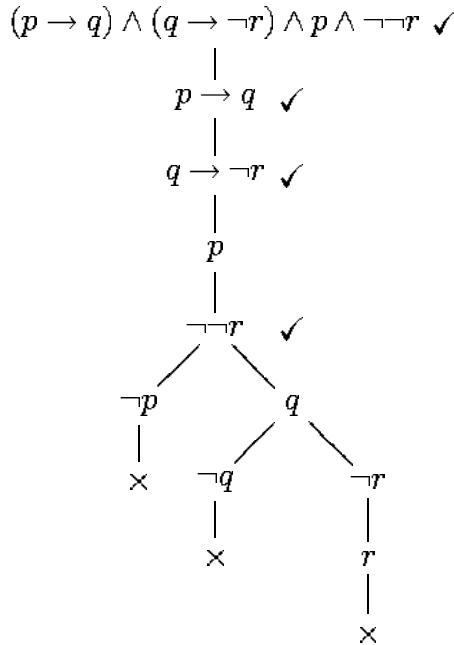


Figura 2.1: Un ejemplo de árbol semántico proposicional.

Cuando nos encontramos con dos fórmulas de LP, digamos ϕ y ψ , éstas (como se puede verificar de forma muy sencilla) podrían tener significado opuesto, es decir, desde el punto de vista semántico podrían ser $\phi = \neg\psi$, pero sintácticamente podrían tener una forma muy distinta. Por ejemplo, tenemos que $\phi = \neg\psi$ donde $\phi = p \rightarrow q$ y $\psi = p \wedge \neg q$. El método de la resolución está basado en el reconocimiento de pares (*fórmula, negación*); por lo tanto, es necesario utilizar reglas adecuadas de transformación que nos permitan llegar a una forma común para las fórmulas de LP en la que estamos interesados. Las reglas más importantes son:

- $\phi \leftrightarrow \psi = \phi \rightarrow \psi \wedge \psi \rightarrow \phi$;
- $\phi \rightarrow \psi = \neg\phi \vee \psi$;
- $\neg(\phi \wedge \psi) = \neg\phi \vee \neg\psi$ (ley de De Morgan);
- $\neg(\phi \vee \psi) = \neg\phi \wedge \neg\psi$ (ley de De Morgan);
- $\neg\neg\phi = \phi$;
- $\phi \wedge (\psi \vee \varphi) = (\phi \wedge \psi) \vee (\phi \wedge \varphi)$;
- $\phi \vee (\psi \wedge \varphi) = (\phi \vee \psi) \wedge (\phi \vee \varphi)$;

Estas reglas, aplicadas de forma iterativa, permiten eliminar de una fórmula proposicional todos los símbolos \leftrightarrow y \rightarrow , poner los símbolos \neg delante de los símbolos proposicionales (los símbolos proposicionales o negaciones de símbolos proposicionales se denominan, en general, *literales* o *átomos*), y llegar a una forma en la cual la conectiva dominante (la más externa) es \wedge , y el resto de fórmulas consiste en una disyunción de literales. Por ejemplo, la fórmula $\neg(p \rightarrow (q \wedge r))$ puede transformarse del siguiente modo:

1. $\neg(p \rightarrow (q \wedge r))$ (fórmula de salida);
2. $\neg(\neg p \vee (q \wedge r))$ (eliminación de \rightarrow);
3. $\neg\neg p \wedge \neg(q \wedge r))$ (ley de De Morgan);
4. $p \wedge \neg(q \wedge r))$ (eliminación de \neg);
5. $p \wedge (\neg q \vee \neg r)$ (ley de De Morgan).

Una disyunción de literales ($\phi_1 \vee \dots \vee \phi_k$) se denomina *cláusula*, y una fórmula expresada como conjunción de cláusulas se dice que está en *forma normal conjuntiva*. Toda fórmula de LP se puede poner en forma normal conjuntiva aplicando las reglas anteriores. Así pues, cuando queramos demostrar que una fórmula es insatisfacible a través de la resolución, primero hay que negar dicha fórmula y después hay que transformarla en conjunción de cláusulas de la misma manera que se hizo en el ejemplo anterior. Por ejemplo, la fórmula anterior ha sido transformada en la conjunción de las cláusulas $\{p\}$ y $\{\neg q \vee \neg r\}$. Por último, hay que aplicar la regla de resolución a dos cláusulas de forma iterativa hasta que obtenemos la cláusula vacía. Por ejemplo, si tenemos, en el conjunto de las cláusulas, dos cláusulas de la forma

$$\{p, q, \neg r\} \quad \{p, \neg q, \neg s\},$$

entonces añadiremos la cláusula

$$\{p, \neg r, \neg s\}.$$

Después de cada paso de resolución, se obtiene una nueva cláusula que se añade al conjunto de cláusulas. Cada una de ellas representa, por así decirlo, una parte de los literales que deben realizarse para que la fórmula sea satisfacible; por ejemplo, si tenemos la fórmula en forma normal conjuntiva

$$\phi = (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg s)$$

podemos decir que, para que ϕ sea satisfacible, por un lado *al menos uno de los literales* $p, q, \neg r$ debe ser verdad (es decir, cualquier modelo de ϕ debe evaluar como verdadero a p , o bien a q , o bien evaluar como falso a r), y por otro lado, *al menos uno de los literales* $\neg p, q, \neg s$ debe ser verdad. Por lo tanto, en cada paso de resolución se eliminan las posibles tautologías (p y $\neg p$), y si después de uno de estos pasos nos encontramos con una cláusula *vacía*, quiere decir que en la fórmula hay una contradicción que

no se puede resolver y, consecuentemente, la fórmula es insatisfacible. El método de resolución completo se puede resumir como se muestra en el Algoritmo 2.2.

Algoritmo 2.2 Un algoritmo de resolución para LP.

```

1: Entrada:  $\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi$ 
2: Transforma la entrada en  $\phi_1 \wedge \dots \wedge \phi_n \wedge \neg\phi$ 
3: Transforma la fórmula obtenida en el conjunto de cláusulas  $\Gamma_1, \dots, \Gamma_k$ , y elimina toda cláusula
   que contiene  $p$  y  $\neg p$ 
4: bucle
5:   si Existen dos cláusulas  $\Gamma_i, \Gamma_j$  que pueden ser utilizadas para aplicar la regla de resolución
      entonces
6:     Aplica la regla de resolución obteniendo  $\Gamma'$ , y añade  $\Gamma'$  al conjunto de cláusulas
7:   fin si
8: fin bucle
9: si En el conjunto de las cláusula aparece la cláusula vacía entonces
10: Devuelve NO SATISFACIBLE
11: fin si
12: si En el conjunto de las cláusula NO aparece la cláusula vacía entonces
13: Devuelve SATISFACIBLE
14: fin si

```

Como ejemplo, consideremos la fórmula

$$\phi = (p \wedge (\neg q \rightarrow \neg p) \wedge (q \rightarrow r)) \rightarrow r.$$

Si queremos demostrar que ϕ es un razonamiento válido, procederemos aplicando el algoritmo de resolución a la fórmula $\neg\phi$, es decir, a la fórmula

$$\phi = (p \wedge (\neg q \rightarrow \neg p) \wedge (q \rightarrow r)) \wedge \neg r.$$

Después de los pasos necesarios de trasformación, obtenemos el siguiente conjunto de cláusulas:

$$\{p\}, \{q, \neg p\}, \{\neg q, r\}, \{\neg r\},$$

al que se puede aplicar la resolución como se muestra en la Figura 2.2.

Optimizar la resolución: cláusulas de Horn. La complejidad de LP se puede mejorar pagando el precio de tener una menor expresividad. Existen aplicaciones de IA para las cuales no es necesario utilizar toda la expresividad de LP. Es posible demostrar (véase, por ejemplo, [Kowalski, 1979]) que si en un conjunto de cláusulas todas tienen una forma llamada de Horn, entonces el problema de la satisfacibilidad tiene una complejidad inferior. Una cláusula de *Horn* es:

- Un átomo.
- Una implicación tal que el antecedente es una conjunción de literales positivos, y el consecuente es un solo literal positivo.
- Una implicación tal que el antecedente es una conjunción de literales negativos, y el consecuente es vacío.

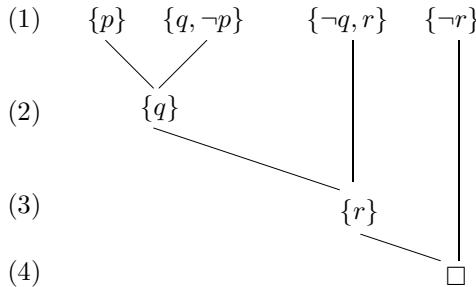


Figura 2.2: Un ejemplo de resolución.

Por ejemplo, $p, p \wedge q \wedge r \rightarrow s$ son cláusulas de Horn, mientras que $p \wedge q$ no lo es. Es posible demostrar que la deducción con cláusulas de Horn es lineal ($O(n)$). Con cláusulas de Horn, el método basado en la resolución puede optimizarse de varias maneras; un ejemplo es el método de *encadenamiento hacia delante (forward chaining)*. En concreto, al tener cláusulas del tipo $p_1 \wedge \dots \wedge p_{k-1} \rightarrow p_k$, cada vez que tenemos en nuestra base de conocimiento el conjunto de *hechos* $\{p_1, \dots, p_{k-1}\}$, podemos deducir el hecho p_k , y añadirlo a la base de conocimiento. Una buena referencia bibliográfica sobre métodos de resolución optimizados es [Padawitz, 1988].

2.3 Lógica de primer orden

En esta sección, nos centraremos en la clásica extensión del lenguaje de LP, es decir, en el lenguaje de la lógica de primer orden o lógica de predicados, que denotaremos como LPO. Después de presentar su sintaxis y su semántica, nos centraremos en su poder expresivo y en los métodos deductivos más importantes. Como referencias bibliográficas recomendamos, como en la sección anterior, [Schechter, 2005].

2.3.1 Sintaxis y semántica

En las secciones anteriores hemos visto cuáles son los límites más importantes de LP. La *lógica de primer orden* es una extensión de LP, en la cual se introducen *variables* para denotar elementos del dominio, *cuantificadores* y *predicados*. Una gramática abstracta para generar fórmulas bien formadas puede ser la siguiente:

$$t ::= a \mid x \mid f(t_1, \dots, t_k),$$

$$\phi ::= p(t_1, \dots, t_n) \mid \neg\phi \mid \phi \wedge \psi \mid \exists x\phi(x),$$

donde el predicado p es de aridad n . Si ϕ es una fórmula de LPO, entonces con $\phi(x_1, \dots, x_n)$ se denota una fórmula que presenta exactamente n variables *libres* (o sea, que no se encuentran en el *ámbito* de algún cuantificador). Los elementos denotados con t_1, \dots, t_k son llamados *términos*, y, como se puede ver en la gramática abstracta que los genera, pueden contener símbolos como f (también g, h, \dots) llamados *functores* o *símbolos funcionales*. La presencia de símbolos funcionales en el lenguaje aumenta el poder expresivo de LPO, y, como veremos, no afecta a sus propiedades computacionales. Más adelante, daremos algunas ideas básicas de las lógicas de ordenes superiores al primero, y veremos cómo los símbolos funcionales se relacionan con los cuantificadores de segundo orden. En la literatura, se le suele denominar *lógica de primer orden* a LPO en cuyo lenguaje *no* aparecen símbolos funcionales, y *lógica de primer orden con funciones* a LPO. Todas las conectivas de LP también forman parte del lenguaje de LPO (se pueden definir exactamente como en el caso de LP). Además, tenemos otro cuantificador *universal*, denotado con \forall , que es equivalente a $\neg\exists\neg$.

Considérese el ejemplo siguiente:

El robot se encuentra con el brazo mecánico vacío. Si el robot está sujetando algo en el brazo mecánico, no puede levantar nada más desde el suelo. Si existe un objeto que se encuentra en el suelo, entonces el robot tiene que levantarla o pasar a su lado. Si el robot encuentra un obstáculo y no puede levantarla, entonces tiene que pasar a su lado, dejar lo que tiene en el brazo en el punto P, y volver a por el objeto que ha encontrado, hasta que todos los objetos están en el punto P.

Mediante LPO, somos capaces de formalizar el anterior párrafo de manera muy sencilla. La propiedad *vacío* del brazo mecánico podría ser indicada con p (un predicado 0-ario, es decir, una proposición). *Levantar* un objeto, o *sujetarlo*, es un predicado unario $q(x)$, donde el parámetro x indica el objeto sujetado. Entonces, tenemos:

$$\exists x(q(x) \rightarrow \neg p).$$

Asimismo, la premisa *si existe un objeto que se encuentra en el suelo, entonces el robot tiene que levantarla o pasar a su lado* se puede formalizar con

$$\exists x(r(x) \rightarrow s(x) \vee t(x)),$$

donde $r(x), s(x), t(x)$ son los respectivos predicados unarios. El lector puede completar este ejemplo resolviendo el Problema 17.

Las fórmulas de LPO se interpretan a partir de un *dominio* D . Los elementos del dominio son los valores que toman las variables x, y, \dots . Además, por cada uno de los símbolos de predicados p^n y funcionales f^m es necesario dar una interpretación \hat{p}^n (es decir, una relación n -aria, subconjunto de $\underbrace{D \times \dots \times D}_n$) y \hat{f}^m (es decir, una función

$\underbrace{D \times \dots \times D}_m \mapsto D$). Formalmente:

- $M \models p^n(t_1, \dots, t_n)$ si y sólo si $(I(t_1), \dots, I(t_n)) \in \hat{p}^n$, donde $I(t_i)$ es la interpretación del término t_i , que depende de la interpretación de las constantes y de las funciones que aparecen en t_i ;
- $M \models \neg\psi$ si y sólo si no es verdad que $M \models \psi$;
- $M \models \psi \wedge \varphi$ si y sólo si $M \models \psi$ y $M \models \varphi$;
- $M \models \exists x\psi(x)$ si y sólo si existe un elemento $d \in D$ tal que $M \models \psi[x/d]$, donde $\psi[x/d]$ indica que todas las ocurrencias libres de x han sido sustituidas por d .

Las constantes que aparecen en los términos se interpretan directamente; por ejemplo, en la fórmula

$$\text{Hombre}(\text{Sócrates})$$

la constante *Sócrates* es un símbolo que viene interpretado como el famoso filósofo Sócrates en un dominio en el que aparecen objetos, personas, animales,...

En el lenguaje de LPO se puede expresar una gran variedad de situaciones. Por ejemplo:

Si el robot tiene la batería cargada, y cumple el conjunto de acciones A, entonces luego tendrá la batería descargada.

En este ejemplo queremos tener en cuenta la componente temporal. Supongamos que modelamos el tiempo como un conjunto de puntos D (nuestro dominio) linealmente ordenado; los predicados p_A (*el robot cumple el conjunto de acciones A*), p_C (*batería cargada*), y p_D (*batería descargada*) indican las situaciones que queremos modelar, y dependen de un parámetro que indica el momento temporal. El problema es que debemos añadir las condiciones para que el dominio sea linealmente ordenado²:

$$\forall x(\neg(x < x)) \tag{2.5}$$

$$\forall x, y, z(x < y \wedge y < z \rightarrow x < z) \tag{2.6}$$

$$\forall x, y(x < y \wedge y < x \rightarrow x = y) \tag{2.7}$$

$$\forall x, y(x < y \vee y < x \vee x = y), \tag{2.8}$$

y, finalmente, podemos formalizar el ejemplo anterior como sigue:

$$(1.5) \wedge (1.6) \wedge (1.7) \wedge (1.8) \wedge \exists x(p_C(x) \wedge \exists y(x < y \wedge p_A(y) \rightarrow \exists z(y < z \wedge p_D(z)))).$$

²A partir de ahora utilizaremos notación infija para expresar las relaciones $<$ e $=$ entre los elementos de un dominio linealmente ordenado.

2.3.2 Poder expresivo y límites de la lógica de primer orden

El gran problema de la IA es *qué* expresar, y no *cómo* expresarlo. La lógica LPO no hace nada más que proporcionar un lenguaje uniforme con el que se puede formalizar el conocimiento sobre la parte de la realidad que nos interesa. Como hemos visto, el primer paso siempre consiste en conceptualizar el conocimiento en términos de objetos, funciones y relaciones. Luego, utilizamos las expresiones y las fórmulas cuyo significado involucra a dichos objetos, funciones y relaciones. Finalmente, nos preguntamos sobre la satisfactibilidad de las fórmulas teniendo en cuenta únicamente los modelos que respetan ciertas características de nuestro interés. En el lenguaje de LPO podemos expresar gran variedad de situaciones de complejidad arbitraria; sin embargo, los límites más evidentes de LPO, aparte de los que veremos en la próxima sección cuando hablemos de extensiones de LPO, consisten en el esfuerzo (que se traduce en complejidad de las fórmulas) que tenemos que hacer para expresar situaciones típicas en IA en las que hay que tener en cuenta relaciones con determinadas características. En el ejemplo de la sección 2.3.1, cuando intentamos modelar un dominio para que tenga características temporales, nos hemos dado cuenta de que expresar las características algebraicas necesarias supone fórmulas relativamente largas; por otra parte, hemos visto que los métodos deductivos dependen en gran parte de la longitud de las fórmulas, por lo que manejar fórmulas complejas constituye un límite que no podemos no tener en cuenta; por último, como hemos dicho anteriormente, se cumple la ecuación: *más poder expresivo = más complejidad computacional* (en casos como LPO también *indecidibilidad*)= *métodos deductivos más complicados*.

2.3.3 Métodos deductivos y coste computacional

El método del árbol semántico. Los métodos de deducción para el lenguaje de LPO son básicamente extensiones de los métodos para el lenguaje de LP [D'Agostino y otros, 1999]. En el caso del método basado en el desarrollo de un árbol semántico para averiguar si cierta fórmula es insatisfacible, debemos de tener en cuenta que, en general no es posible dar una metodología de decisión para LPO que termine en todos los casos. De hecho LPO es una lógica para la cual el problema de la satisfactibilidad es *indecidable*, lo que significa que no existe (ni puede existir) ningún algoritmo que cumpla las condiciones: (i) es *correcto* (nunca devuelve respuestas equivocadas), (ii) es *completo* (devuelve todas las respuestas correctas), (iii) es *terminante* (nunca entra en un bucle infinito). Los límites de las metodologías para LPO están en el carácter no finito de los dominios; de hecho, no es difícil escribir una fórmula en el lenguaje de LPO que sólo admite modelos infinitos; supongamos por ejemplo que el predicado $<$ ha sido formalizado sobre un orden lineal en el dominio (véase el último ejemplo de la sección 2.3.1), y consideremos la fórmula:

$$\exists x(p(x)) \wedge \forall x(p(x) \rightarrow \exists y(x < y \wedge p(y))). \quad (2.9)$$

Se ve claramente que para satisfacer la fórmula anterior es necesario tener un dominio infinito. A pesar de esta limitación en los algoritmos de deducción, el problema de la satisfactibilidad de la lógica LPO se encuentra en una clase de problemas

‘no demasiado complicados’; en otras palabras, existen algoritmos (como los que veremos) para LPO que son correctos y completos, aunque no siempre terminan. Esta clase de problemas se denominan *semidecidibles* (o *recursivamente enumerables*); lógicas más expresivas de LPO, como por ejemplo las de órdenes superiores, son tales que sus problemas de satisfactibilidad son aún más complejos, y no es posible encontrar si quiera algoritmos completos.

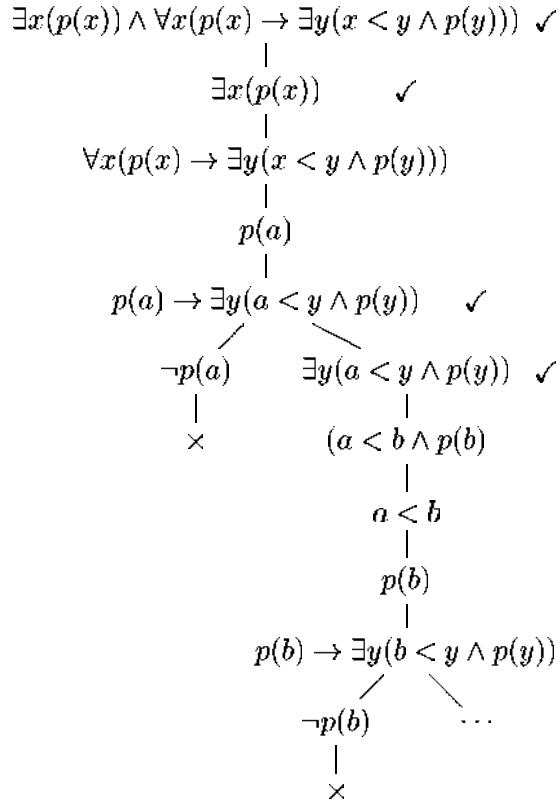


Figura 2.3: Un ejemplo de árbol semántico de primer orden.

El método del árbol semántico para el lenguaje LPO contiene las mismas reglas que en el caso de LP, más las siguientes reglas precisas para los cuantificadores:

- Cuantificador *universal*, es decir, $\forall x\phi(x)$: en este caso, para todas las constantes a que han sido utilizadas en la rama considerada evaluaremos la fórmula $\phi(a)$, teniendo en cuenta que la fórmula $\forall x(p(x))$ permanece activa.
- Cuantificador *existencial*, es decir, $\exists x\phi(x)$: en este caso, es preciso introducir una constante a nueva (que no haya sido utilizada en ninguna rama abierta hasta el momento), y evaluar la fórmula $\phi(a)$.

Como se puede ver, al evaluar la satisfacibilidad de una fórmula de LPO, se construye un dominio D . Si evaluamos la satisfacibilidad de (2.9) como en la Figura 2.3, el primer cuantificador existencial nos obliga a poner una constante a en el dominio D , y evaluar la fórmula $p(a)$; más adelante, en la única rama que tenemos, el cuantificador universal nos obliga a evaluar la fórmula $p(a) \rightarrow \exists y(a < y \wedge p(y))$. Luego, la rama que tiene en cuenta la posibilidad $\neg p(a)$, se cierra inmediatamente por contradicción, lo que implica seguir con la rama que tiene en cuenta la posibilidad $\exists y(a < y \wedge p(y))$. La regla del existencial se aplica otra vez, con la introducción de una nueva constante b , que sustituye la variable y . Sin embargo, al estar la fórmula $\forall x(p(x) \rightarrow \exists y(x < y \wedge p(y)))$ siempre activa, la única rama activa no puede llegar a una contradicción y, al mismo tiempo, llegar a una situación en la que el algoritmo termine. Por lo tanto, este es un caso en el que no somos capaces de decir si la fórmula es o no satisfacible.

El método de resolución. En el caso de LPO el método de resolución es un poco más complicado que en el caso de LP. Básicamente hay dos problemas nuevos con respecto al caso proposicional:

1. Los cuantificadores en la fase de transformación de la fórmula de salida en forma clausal.
2. El reconocimiento de las fórmulas atómicas contradictorias, cuando éstas contengan variables o términos.

Empezamos con la transformación de ϕ en forma clausal. El algoritmo que permite esta transformación debe tener en cuenta tanto las reglas utilizadas en el caso de LPO, más algunas reglas que permiten poner todos los cuantificadores en la parte izquierda de la fórmula. Una fórmula que tiene esta estructura se dice que está en forma *prenexa*. Se puede demostrar que, para toda fórmula ϕ de LPO, existe una fórmula equivalente ϕ' de LPO en forma prenexa. Suponiendo que, para cierta fórmula ϕ de LPO ya hemos aplicado las reglas proposicionales que permiten eliminar todas las ocurrencias de los operadores \rightarrow y \leftrightarrow , y poner todos los operadores \neg delante de las fórmulas atómicas, las reglas que permiten poner ϕ en forma prenexa son las siguientes:

- $\psi \circ Qx\varphi(x) = Qx(\psi \circ \varphi(x))$ (donde x no aparece libre en ψ);
- $\psi \circ Qx\varphi(x) = Qy(\psi \circ \varphi(y))$ (donde x aparece libre en ψ , y la variable y es nueva),

donde $\circ \in \{\wedge, \vee\}$, y $Q \in \{\forall, \exists\}$. Veamos un ejemplo; consideremos la fórmula $\neg(\exists x(p(x) \rightarrow \forall y(q(x, y))))$:

1. $\neg(\exists x(\neg p(x) \vee \forall y(q(x, y))))$ (eliminación de \rightarrow);
2. $(\forall x(\neg p(x) \vee \forall y(q(x, y))))$ (interdefinición de \forall y \exists);
3. $(\forall x(\neg\neg p(x) \wedge \neg\forall y(q(x, y))))$ (ley de De Morgan);
4. $(\forall x(p(x) \wedge \exists y\neg(q(x, y))))$ (eliminación de $\neg\neg$, e interdefinición de \forall y \exists);
5. $\forall x\exists y(p(x) \wedge \neg(q(x, y)))$ (cuantificador \exists).

Esencialmente, lo que la metodología de la resolución hace es demostrar que una fórmula no es satisfactible a través de un dominio muy peculiar, llamado *dominio de Herbrand*, que, se puede demostrar, es suficiente para comprobar que la fórmula considerada no es satisfacible bajo ninguna interpretación. La interpretación basada en un dominio de Herbrand es una interpretación cuyo dominio es puramente simbólico; este dominio se obtiene utilizando la(s) constante(s) del lenguaje (si no existe ninguna constante, se introduce una de forma arbitraria) y todos los términos obtenidos utilizando los símbolos funcionales (si existen) y las constantes. En el caso de la fórmula anterior, por ejemplo, vemos que no existen símbolos funcionales, y que, una vez que la fórmula está en forma prenexa, solamente aparece un cuantificador existencial. Esto significa que el dominio de Herbrand está constituido por una sola constante, que, como en el caso del método basado en árboles semánticos, debemos de introducir de forma artificial. Como otro ejemplo, si el lenguaje de una fórmula contiene la constante a y el símbolo funcional unario f , el dominio de Herbrand sería el conjunto infinito $\{a, f(a), f(f(a)), \dots\}$. Para aplicar la resolución, una vez que la fórmula está en forma prenexa, hay que eliminar los cuantificadores, quedando en una forma conocida como *forma normal de Skolem*; este último paso es correcto para decidir la insatisfacibilidad, lo que significa que, si la fórmula inicial es insatisfacible, también lo es su correspondiente fórmula en forma normal de Skolem. Para pasar a forma normal de Skolem hay que aplicar de forma recursiva las siguientes reglas:

- Trasformar $\forall x\phi(x)$ en $\phi(x)$.
- Trasformar $\exists x\phi(x)$ en $\phi(a)$ si no hay cuantificadores \forall a la izquierda de $\exists x$, o en $\phi(f(x_1, \dots, x_n))$ si los cuantificadores $\forall x_1, \dots, \forall x_n$ están a la izquierda de $\exists x$, siendo a una constante nueva en el primer caso y f es un símbolo funcional nuevo en el segundo.

Por ejemplo, en la fórmula anterior encontramos el siguiente grupo de cuantificadores $\forall x\exists y(\phi(x, y))$. Aplicando las reglas que hemos visto, el cuantificador existencial se encuentra a la derecha de un cuantificador universal; esto significa que no podemos elegir libremente una constante para sustituir a la variable y , sino que esta elección depende de x , y por lo tanto, debemos de introducir una función f nueva. Al no haber constantes en el lenguaje, introduciríamos una constante arbitraria a , y la fórmula final quedaría así: $p(x) \wedge \neg q(x, f(x))$.

Con el fin de utilizar un método de refutación como la resolución es importante destacar que la forma normal de Skolem de una fórmula de primer orden es *equisatisfacible* a la fórmula original (es decir, es satisfacible si y sólo si lo es la fórmula original, pero no tiene necesariamente el mismo conjunto de modelos).

En este punto, hemos obtenido una fórmula que puede considerarse como un conjunto de cláusulas, exactamente como en el caso de LP. El último problema que encontramos es el siguiente: supongamos que en una cláusula Γ_1 aparece la fórmula atómica $p(x)$, y en otra cláusula Γ_2 la fórmula atómica $\neg p(a)$; ¿podemos decir que el primero es la negación del segundo? Al ser la variable x cuantificada universalmente, el primer término dice que la propiedad p es cierta para todo elemento del dominio, lo que incluye también la constante a , y, por lo tanto, sí podemos decir que son

una contradicción. Este problema de reconocimiento de términos se conoce como el problema de la *unificación*. En IA, necesitamos un algoritmo implementable para la solución de cada problema; el algoritmo para la unificación de dos términos (que nos dice si dos términos son unificables o no) lo podemos ver en el Algoritmo 2.3. Como ejemplo, consideremos las siguientes fórmulas atómicas: $p(x, a, f(z, w))$ y $\neg p(y, a, f(b, c))$. Al ser dos predicados de aridad tres, debemos preguntarnos si los pares de términos (x, y) , (a, a) , y $(f(z, w), f(b, c))$ son unificables. Utilizando el algoritmo, tenemos que la ecuación $x = y$ nos devuelve SI, la ecuación $a = a$ también, y la ecuación $f(z, w) = f(b, c)$ viene sustituida por las ecuaciones $z = b$, $w = c$, las cuales también nos devuelven SI. Finalmente, el algoritmo de resolución para LPO se puede ver en Algoritmo 2.4.

Algoritmo 2.3 Unificación de términos.

```

1: Entrada: dos cláusulas  $\Gamma_1$  y  $\Gamma_2$ , y dos fórmulas atómicas  $p(t_1, \dots, t_n) \in \Gamma_1$ ,  $\neg p(s_1, \dots, s_n) \in \Gamma_2$ 
2:
3: si  $n \neq m$  entonces
4:   Devuelve NO y termina
5: fin si
6: Introduce las ecuaciones  $t_1 = s_1, \dots, t_n = s_m$ 
7: bucle
8:   si Existe una ecuación  $t_i = s_j$  que no ha sido utilizada todavía entonces
9:     si  $(t_i = f(u_1, \dots, u_k))$  y  $s_j = g(v_1, \dots, v_h)$  o  $(k \neq h)$  o  $(t_i = x \text{ y } s_j \text{ es un término donde aparece } x)$  entonces
10:    Devuelve NO y termina
11:   fin si
12:   si  $t_i = f(u_1, \dots, u_k)$  y  $s_j = f(v_1, \dots, v_h)$  y  $k = h$  entonces
13:     introduce las ecuaciones  $u_1 = v_1, u_k = v_h$ 
14:   fin si
15:   si  $t_i = x \text{ y } s_j = x$ , o  $t_i = a \text{ y } s_j = a$  entonces
16:     elimina la ecuación
17:   fin si
18:   si  $t_i = x \text{ y } s_j$  es un término donde no aparece  $x$  entonces
19:     Aplica la sustitución  $x = s_j$  en la cláusula  $\Gamma$ , en cualquier literal donde aparece el término  $t_i$ 
20:   fin si
21:   si  $t_i = x$  es un término y  $s_j = y$  entonces
22:     elimina la ecuación y añade la ecuación  $s_j = t_i$  al conjunto
23:   fin si
24: fin si
25: fin bucle
26: si hemos considerado todas las ecuaciones, aplicado todas las sustituciones y no hemos contestado negativamente entonces
27:   Devuelve SI
28: fin si
```

Consideremos, por ejemplo, la fórmula

$$(\forall x p(x) \wedge \forall y (p(y) \rightarrow q(y)) \wedge \forall z (q(z) \rightarrow r(z))) \rightarrow r(a).$$

Aplicando el algoritmo, la fórmula de salida se trasforma en $\forall x p(x) \wedge \forall y (p(y) \rightarrow q(y)) \wedge \forall z (q(z) \rightarrow r(z)) \wedge \neg r(a)$, lo que corresponde a las cláusulas $\Gamma_1 = \{p(x)\}$, $\Gamma_2 = \{\neg p(y), q(y)\}$, $\Gamma_3 = \{\neg q(z), r(z)\}$, $\Gamma_4 = \{\neg r(a)\}$; el desarrollo del algoritmo basado en la resolución para este ejemplo se muestra en la Figura 2.4.

Referencias sobre los métodos de resolución en LPO se pueden encontrar en muchos manuales de lógica, como por ejemplo en [Schechter, 2005].

Algoritmo 2.4 El algoritmo de resolución.

- 1: Entrada: $\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi$
 - 2: Trasforma la fórmula de salida en forma prenexa
 - 3: Trasforma la fórmula en forma de Skolem, y representala en forma clausal $\Gamma_1, \dots, \Gamma_n$, eliminando todas las cláusulas que contienen $p(t_1, \dots, t_n)$ y $\neg p(s_1, \dots, s_n)$
 - 4: **bucle**
 - 5: **si** Existen dos cláusulas Γ_i, Γ_j tales que contienen dos predicados $p(t_1, \dots, t_n)$ y $\neg p(s_1, \dots, s_n)$, y t_1, s_1 (resp., t_n, s_n) son términos unificables, (o sea, tales que, después de haber aplicado la sustitución correspondiente permiten aplicar la regla de resolución) **entonces**
 - 6: Aplica la sustitución correspondiente a Γ_i, Γ_j
 - 7: Obtén la nueva cláusula resultante Γ , y añádela al conjunto de cláusulas
 - 8: **fin si**
 - 9: **fin bucle**
 - 10: **si** Una de las cláusula es vacía **entonces**
 - 11: Devuelve NO SATISFACIBLE
 - 12: **fin si**
-

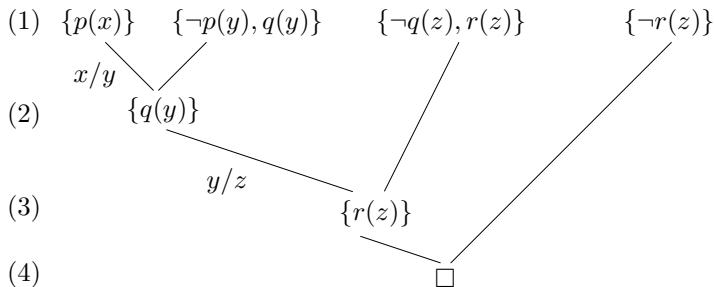


Figura 2.4: Un ejemplo de resolución de primer orden.

2.3.4 Lógicas de orden superior al primero

La lógica de primer orden se considera muchas veces como un lenguaje de referencia con respecto al poder expresivo. En IA, en las últimas décadas se han hecho esfuerzos por encontrar extensiones de la lógica clásica proposicional que presenten características típicas de la lógica de primer orden, pero cuyos lenguajes tengan límites que permitan mantener decidible el problema de la satisfacibilidad. Las lógicas modales y temporales, como veremos, son ejemplos de este tipo. Por otra parte, en lógica y en filosofía se han estudiado lenguajes basados en LPO que poseen un gran poder expresivo y al mismo tiempo una complejidad extremadamente alta. Volvamos otra vez al ejemplo del robot y consideremos la siguiente situación:

Tenemos dos grupos de objetos, denominados A y B. En el estado inicial, los dos grupos pueden tener un número cualquiera de objetos cada uno. El brazo mecánico puede, según se necesite, coger un objeto del grupo A y ponerlo en el grupo B, o al contrario (pero sólo se puede mover un objeto a la vez). La tarea consiste en llegar a tener, mediante sucesivos movimientos, la misma cantidad de objetos en los dos grupos (o una unidad menos, si el número total es impar).

En el momento en que empezamos a formalizar en LPO este párrafo, nos damos cuenta de que no tenemos suficiente poder expresivo para comparar dos cantidades. Para poder hacer esto, necesitamos *variables sobre conjuntos de objetos*, poder *cuantificar* sobre ellas y un símbolo \in que indique la *pertenencia* de un objeto a un conjunto. En una lógica de *segundo orden*, sobre un dominio D , se puede pensar en utilizar tanto cuantificadores sobre elementos del dominio (al igual que en LPO):

$$\forall x\phi(x)$$

como sobre conjuntos de elementos:

$$\forall X\phi(X)$$

La posibilidad de cuantificar sobre un conjunto de elementos (o sobre un conjunto de pares, de ternas, y así sucesivamente) nos permite por ejemplo asegurar la existencia de una función. Por lo tanto, las características típicas de los órdenes superiores al primero, también aparecen ‘disfrazadas’, por así decirlo, en problemas de primer orden; de hecho, preguntarse si una fórmula de primer orden, como por ejemplo

$$\forall x\exists y(p(x, f(y)))$$

es satisfacible, es lo mismo que preguntarse si es satisfacible la fórmula

$$\exists p_f \forall x \exists y \exists z (p_f(y, z) \wedge p(x, z) \wedge \forall x, y, z (p_f(x, y) \wedge p_f(x, z) \rightarrow y = z))$$

donde hemos cuantificado sobre un predicado p_f que de alguna manera representa el functor f , y tiene las características de una función unaria definida en el dominio. Con una lógica de segundo orden, por ejemplo, es posible contar los elementos en un conjunto con respecto a un dominio de primer orden, o comparar dos conjuntos. Por ejemplo, el predicado $E(X, Y)$ (X e Y tienen el mismo número de objetos), donde X, Y son variables de segundo orden, se puede formalizar así:

$$\begin{aligned} E(X, Y) \leftrightarrow & \exists f, g (\forall x (x \in X \rightarrow \exists y (y \in Y \wedge f(x) = y))) \\ & \wedge (\forall y (y \in Y \rightarrow \exists x (x \in X \wedge g(y) = x))) \\ & \wedge \forall x, y (x \neq y \rightarrow f(x) \neq f(y)) \wedge \forall x, y (x \neq y \rightarrow g(x) \neq g(y))), \end{aligned}$$

es decir, dos conjuntos X e Y tienen el mismo número de elementos si y sólo si es posible encontrar dos funciones totales inyectivas (o un isomorfismo) entre ellas.

Las lógicas de órdenes superiores al primero son extremadamente complejas, y en muchos casos no solamente no existen algoritmos para averiguar si una fórmula es válida (*semidecidibilidad*), sino que tampoco es posible averiguar si una fórmula dada es satisfacible; en este caso, se dice que una lógica es *no recursivamente enumerable*. La referencia más indicada para los problemas de decidibilidad/indecidibilidad para lógicas de orden superior al primero es [E.Borger y otros, 1988].

2.3.5 Fragmentos de LPO

Finalmente, con respecto al lenguaje de LPO es posible identificar algunos subconjuntos (de carácter sintáctico) que permiten mejorar las propiedades computacionales. Cuando consideramos una gramática abstracta para un cierto lenguaje L que resulta ser un subconjunto de la gramática para un lenguaje L' , entonces denominamos L un *fragmento* (sintáctico) de L' . Ejemplos en este sentido son:

- Lógicas con un número limitado de variables.
- Fragmentos con *guardia*.
- Fragmentos (tanto de LPO como de la lógica de segundo orden) en los que se limita la aridad máxima de los predicados.

Las lógicas de primer orden con un número limitado de variables son básicamente lógicas de primer orden en las que se permiten utilizar solamente algunos símbolos de variable, los cuales se pueden reutilizar un número arbitrario de veces. Por ejemplo, en el lenguaje LPO_2 , sólo se permiten utilizar dos variables en cada fórmula. Algunos trabajos como [Immerman y Kozen, 1987; Scott, 1962], por ejemplo, demuestran cómo podemos medir el poder expresivo de lógicas limitadas en el número de variables, que, en algunos casos, resultan ser más expresivas de lo que puede parecer a primera vista. Algunos estudios teóricos se han centrado en contestar a preguntas como *¿cuántas variables son necesarias para expresar la propiedad P ?* *¿Qué propiedades se pueden expresar con N variables?* El estudio de lógicas con un número limitado de variables es de gran importancia ya que nos sirve para buscar el límite entre la decidibilidad y la indecibilidad de las lógicas de primer orden.

Los fragmentos con guardia, por otra parte, han sido propuestos como resultado de los estudios de decidibilidad de las lógicas modales (véase la siguiente sección). Un fragmento de LPO se define *con guardia* si la cuantificación de las variables está limitada por alguna relación. Por ejemplo, en un fragmento con guardia la fórmula

$$\forall x\phi(x)$$

no se admitiría; en su lugar, se supone la presencia de una relación, digamos, binaria, R , y se cuantifica así:

$$\forall x(xRy \rightarrow \phi(x))$$

Intuitivamente, las cuantificaciones con guardia no permiten a los elementos del dominio estar ‘en cualquier sitio’, sino que les obligan a seguir cierta estructura. Las características de las guardias (que pueden ser simples relaciones, o fórmulas proposicionales en alguna forma determinada) determinan las propiedades computacionales del fragmento que se considera. Véanse [E.Borger y otros, 1988; Vardi, 1997] como referencias bibliográficas acerca de los resultados más recientes sobre fragmentos con guardias.

Por último, del mismo modo que las lógicas modales han sugerido el estudio, en general, de fragmentos de primer orden con guardias, las lógicas temporales (véase

la sección 2.5), encajan en un marco más general de lógicas, de primer y segundo orden, en las que se limita la aridad máxima de los predicados. Por ejemplo, la Lógica Monádica de Primer Orden, denotada con MFO es exactamente como LPO pero con la limitación de que todo predicado sólo puede ser unario; es decir, en las fórmulas del lenguaje el elemento sintáctico $p(x)$ está permitido, pero no $p(x, y)$. Estas lógicas encuentran muchas aplicaciones y despiertan un gran interés, sobre todo cuando son interpretadas en estructuras como órdenes lineales. Existen varias referencias acerca de este argumento; un ejemplo es [Frick y Grohe, 2002].

2.4 Extensiones de las lógicas clásicas

2.4.1 ¿Por qué extender las lógicas clásicas?

Podemos resumir las conclusiones de las secciones anteriores como sigue: *el lenguaje LP es muy poco expresivo, pero tiene buenas propiedades computacionales y sus métodos deductivos son fácilmente implementables; por otro lado, prácticamente todo lo que podemos expresar con respecto a los razonamientos de un sistema inteligente encuentra su formalización en el lenguaje LPO, cuyos sistemas deductivos tienen malas propiedades computacionales.* Pero, ¿cuáles son exactamente las características que nos gustaría poder expresar? Podríamos enumerar algunas de ellas:

- Capacidad de expresar situaciones hipotéticas, mundos (realidades) posibles, relacionadas de alguna forma.
- Capacidad de expresar situaciones de carácter temporal.
- Capacidad de expresar situaciones de carácter espacial.

Entonces, una buena pregunta es la siguiente: ¿es posible alcanzar dichos objetivos utilizando alguna extensión del lenguaje de LP pero que no suponga utilizar todo el poder expresivo de LPO (que, por lo visto, es la fuente de la indecidibilidad de LPO)? Las lógicas modales permiten expresar, de forma sencilla, situaciones hipotéticas y mundos posibles, y, en alguna de sus especializaciones, también situaciones de carácter espacio/temporal, como veremos en esta sección y en la siguiente. Una referencia bibliográfica muy reciente y completa es [Blackburn y otros, 2002].

2.4.2 Lógicas no monotónicas, razonamiento del sentido común y otras consideraciones

Antes de examinar teorías no propiamente clásicas como las teorías modales y temporales, cabe destacar que la IA como disciplina ha levantado muchas cuestiones de carácter filosófico acerca de la naturaleza del razonamiento. Si por un lado las teorías clásicas y sus extensiones han encontrado numerosas aplicaciones y se han desarrollado metodológicas muy eficientes para ellas, por el otro lado muchos autores ponen en duda, con respecto a las posibilidades de construir agentes inteligentes,

algunas bases sobre las que la lógica clásica se construye. En este apartado, nos limitaremos a reportar algunas de estas consideraciones.

En primer lugar, algunos autores observan que la lógica clásica y sus derivaciones no son adecuadas para el tratamiento de una parte de los razonamientos prácticos que interesan en la IA. Algunos estudios evidencian que los procesos de razonamientos utilizados por el hombre están muy alejados de los de la lógica matemática, y que en general el hombre (el *agente inteligente* por definición), es mucho más eficaz a la hora resolver problemas de carácter práctico (*si tengo 10 euros y compro tres pares de calcetines por un euro y medio cada uno, ¿cuánto recibiré de cambio?*) antes que problemas de carácter más abstractos (*cuanto suma $10+(1.5*3)$?*). Podemos citar algunos argumentos: se ha observado en primer lugar que las cantidades matemáticas son ‘definidas’, mientras la mayoría de las cantidades utilizadas por una mente humana son ‘indefinidas’ sobre un dominio continuo (y no discreto), y presentan errores y ruido; en segundo lugar, cada idea y acción depende de un contexto, el cual puede ser difícil de tener cuenta en un proceso automático; tercero, según autores como Wittgenstein (1958), los fenómenos no pueden siempre ser descompuestos en términos elementales, sino que tienen que ser tratados como un elemento único; cuarto, en la lógica clásica no podemos tener en cuenta la intencionalidad de una acción, que normalmente es importante a la hora de tomar decisiones.

Las lógicas modales, y en particular las lógicas temporales y espaciales, constituyen intentos (que han tenido un éxito importante) para la resolución de algunos de estos problemas. Por otra parte, la clase de lógicas y metodologías llamadas *no monótonas* constituye una rama importante en el contexto del razonamiento automático. Resumiendo, una lógica se dice no monótona cuando es capaz, de alguna manera, de volver atrás en sus conclusiones. De hecho, en el mundo real, donde nos regimos por el sentido común, algunas conclusiones son ciertas hasta que una nueva información nos hace cambiar de idea, y pone en duda toda la línea de razonamiento seguida hasta el momento. Dicho de otra forma, las creencias o conclusiones son *derrotables* o *anulables*. En la lógica clásica este tipo de razonamiento no puede ser expresado, siendo la lógica clásica *monótona*, es decir, permite solamente alcanzar, a través de la deducción, nuevas verdades. Intuitivamente, la propiedad de monotonicidad de la lógica clásica nos dice que el hecho de aprender algo nuevo, no puede reducir el conjunto de cosas que ya sabemos. Por el contrario, en una lógica no monótona, la verdad de una proposición puede cambiar según vayan apareciendo nuevos axiomas. Mientras que en una lógica clásica temporal o espacial este comportamiento está regulado por alguna estructura fija (como un orden lineal), en una lógica no monótona no tenemos esta limitación. La lógica clásica (en todas sus formas) utiliza reglas del tipo: *si p es un teorema, entonces q es un teorema*; en una lógica no monótona, una regla de inferencia puede ser del tipo *q es un teorema si p no es un teorema*.

La literatura en IA contiene numerosas referencias que tratan de sistemas no clásicos para el razonamiento automático. Dentro del grupo de las lógicas no monótonas podemos mencionar, entre otras, la Lógica por Defecto [Reiter, 1980], donde se pueden expresar reglas del tipo *si p es cierto pero desconocemos q entonces s es cierto*, es decir, se establece una conclusión por defecto (*s*) y una excepción (*q*); la lógica clásica con la Hipótesis del Mundo Cerrado [Reiter, 1978], que consiste básicamente

en suponer como falso todo aquello que no está explícitamente afirmado; la lógica Autoepistémica o la Circunscripción de predicados [McCarthy, 1987]. Otras extensiones de la lógica clásica que merecen la pena nombrar, sin entrar en detalles, son las *lógicas multivaluadas*, las cuales, se caracterizan por permitir más de dos valores de verdad (como sabemos la lógica clásica sólo admite los valores verdadero y falso); un tipo especial de lógica multivaluada es la lógica *borrosa* o *difusa* donde las proposiciones tienen un grado de verdad que se asigna mediante una función de pertenencia que toma valores en el intervalo real $[0, 1]$. Por último, mencionamos la *lógica intuicionista* cuya sintaxis es la misma que la de la lógica proposicional o de primer orden, con la principal diferencia de que en esta lógica algunas fórmulas como $\phi \vee \neg\phi$ o $\neg\neg\phi \rightarrow \phi$ no son tautologías. Para entrar en más detalles sobre estos y otros tipos de lógicas no clásicas el lector interesado puede consultar, entre otros, los textos [Gabbay y otros, 1994; Priest, 2001].

2.4.3 Lógicas modales y mundos posibles

Como hemos visto en la sección 2.2.1, un modelo (o mundo) de una fórmula o un conjunto de fórmulas en el lenguaje LP es una evaluación de los símbolos proposicionales que aparecen en la fórmula o en el conjunto de fórmulas. Supongamos ahora que disponemos de varios mundos, digamos $W = \{w_1, w_2, \dots\}$, y de una *relación* R que permite, según sus propiedades, pasar de un mundo w_i a otro mundo w_j . En este caso, podríamos expresar una situación como esta: *si en el mundo actual p es válida y q no lo es, entonces, existe otro mundo relacionado con el mundo actual, en el que q es válida y p no lo es*. En lógica de primer orden esta situación se expresaría así:

$$p(x) \wedge \neg q(x) \rightarrow \exists y(r(x, y) \wedge q(y) \wedge \neg p(y))$$

donde r representa la relación R , y x, y representan los mundos posibles. Pero si pudiésemos expresar de alguna forma los conceptos *existe un mundo ϕ* y su opuesto *por cualquier mundo ϕ* , no necesitaríamos utilizar todo el poder de LPO.

Imaginemos una gramática formal extendida para LP, como la siguiente:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \Diamond\phi$$

donde p es un símbolo proposicional, y \Diamond es el cuantificador existencial que permite desplazarse entre los mundos. Un *modelo* del lenguaje de la lógica modal (LM) es una terna $M = (W, R, V)$, donde W es un conjunto de mundos (o modelos de LP), R es una relación binaria $R \subseteq W \times W$, y V es una función de evaluación que, por cada uno de los mundos, nos dice cuáles son los símbolos proposicionales que se satisfacen en ellos. Consideramos la Figura 2.5. El modelo representado tiene tres mundos, w_1, w_2 y w_3 , y es un modelo para un lenguaje en el que sólo figuran dos símbolos proposicionales, es decir, p y q . Vemos que en el mundo w_1 ambas, p y q , se realizan, en el mundo w_2 se realiza q pero no p , mientras que en el mundo w_3 se realiza p pero no q . Además, sabemos que desde w_1 podemos llegar a w_2 y w_3 , pero, por ejemplo, desde w_2 no podemos llegar a w_1 .

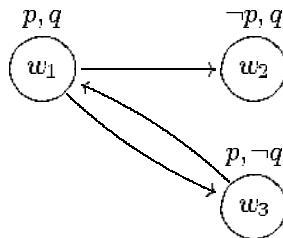


Figura 2.5: Un modelo modal.

La satisfacibilidad de una fórmula ϕ en un modelo modal M , depende de una serie de parámetros:

1. Las propiedades de la relación R .
2. El mundo en que evaluamos la fórmula.

Es decir, la misma fórmula puede ser evaluada a verdadero en un modelo M y en un mundo w_i , y a falso en el mismo modelo pero en otro mundo w_j . Formalmente:

- $M, w_i \Vdash \Diamond\phi$ si y sólo si existe w_j tal que $R(w_i, w_j)$ y $M, w_j \Vdash \phi$.

Nombre	Propiedad
K	No hay condiciones sobre la relación R
K4	R es transitiva, es decir, $\forall x, y, z(r(x, y) \wedge r(y, z) \rightarrow r(x, z))$
T	R es reflexiva, es decir, $\forall x R(x, x)$
B	R es simétrica, es decir, $\forall x, y(R(x, y) \leftrightarrow R(y, x))$

Tabla 2.2: Algunas lógicas modales.

Para expresar una propiedad que es cierta para todos los mundos que son alcanzables a partir del mundo actual, utilizamos el símbolo $\Box = \neg\Diamond\neg$. En un lenguaje modal, decimos que una fórmula ϕ es *satisfacible* si y sólo si existe un modelo modal M y un mundo w_i tales que $M, w_i \Vdash \phi$. Asimismo, decimos que ϕ es *válida en un modelo M* si y sólo si $M, w_i \Vdash \phi$ para cualquier mundo w_i , y *válida* si y sólo si es válida en todos los modelos modales posibles. Las propiedades computacionales de una lógica modal dependen directamente de las propiedades de la relación R . Existen varias lógicas modales, que han sido aplicadas en varios contextos, que se pueden clasificar según las propiedades de R ; algunos ejemplos de lógicas modales pueden verse en la Tabla 2.2.

El lenguaje de LM ha sido interpretado de diferentes maneras, cada una de las cuales tiene diferentes aplicaciones. En particular, hay tres interpretaciones que se pueden considerar, desde un punto de vista histórico, las más importantes. Primero, el símbolo \Diamond puede interpretarse como *possible*: en este caso, la fórmula $\Diamond\phi$ se lee ϕ es

possible, y la fórmula $\Box\phi$ se lee ϕ es necesario. En esta interpretación, podemos expresar conceptos como *todo lo que es necesario es posible*, con la fórmula $\Box\phi \rightarrow \Diamond\phi$ (o sea, requiriendo que la fórmula sea válida). Segundo, en la lógica *epistémica*, la fórmula $\Box\phi$ se interpreta como *el agente (inteligente) conoce ϕ* . Esta interpretación se utiliza para la representación del conocimiento. Tercero, en la lógica de la *demostrabilidad*, la fórmula $\Box\phi$ se interpreta como ϕ es demostrable (en cierta teoría aritmética), y se utiliza en la búsqueda de axiomatizaciones completas de teorías aritméticas (pero no tiene aplicaciones importantes en IA).

2.4.4 Métodos deductivos y coste computacional de la Lógica Modal

Al existir muchas variantes de LM que dependen de las propiedades de la relación de accesibilidad, y también otras variantes del lenguaje básico que incluyen otros operadores modales, también existen varios métodos deductivos. Aquí, nos centraremos en la lógica más sencilla, es decir, la lógica K, y veremos solamente un método deductivo para K que es la extensión del método basado en árboles semánticos para la lógica LP.

Árboles semánticos para K. La metodología basada en árboles semánticos para la lógica K es bastante sencilla, y, como hemos dicho, está basada esencialmente en la misma metodología para LP. Antes de poder aplicar algún método de razonamiento, con el fin de disminuir el número total de las reglas, el primer paso siempre consiste en poner la fórmula considerada en una forma más sencilla, pero equivalente; lo que queremos obtener es una forma que tenga los operadores \neg sólo delante de símbolos proposicionales. Las reglas para obtener la forma normal de ϕ (llamada *negated normal form*, o NNF), son las mismas que vimos en el caso de la resolución para LP, más las dos reglas siguientes:

- transforma $\neg\Diamond\phi$ en $\Box\neg\phi$;
- transforma $\neg\Box\phi$ en $\Diamond\neg\phi$.

Por ejemplo, si tenemos la fórmula $\neg\Box(p \wedge \neg\Diamond q)$, podemos aplicar las reglas que hemos visto, obteniendo $\Diamond\neg(p \wedge \Box\neg q)$, y, con otro paso, $\Diamond(\neg p \vee \neg\Box\neg q)$, y, finalmente, $\Diamond(\neg p \vee \Diamond q)$.

En este punto, tenemos que distinguir entre dos clases de reglas para el desarrollo de un árbol semántico: reglas proposicionales (que serán prácticamente idénticas al caso de LP), y reglas modales. Exactamente como en el caso de LP, una rama quedará cerrada en cuanto se encuentre un símbolo proposicional y su negación; la diferencia está en que en este caso, esta contradicción habrá que encontrarla en el mismo mundo de evaluación. La regla que permitirá el desarrollo de un caso como $\Diamond\phi$ será la encargada de construir nuevos mundos de evaluación, mientras que la regla para el caso $\Box\phi$ aplicará la evaluación de ϕ a todos los mundos accesibles a partir del mundo actual. Hay que tener en cuenta que la lógica K no aplica ninguna restricción a la relación R , lo que significa que, por ejemplo, a partir de un mundo w_i es posible acceder a

través de \Diamond también al mismo mundo w_i . También, debemos tener en cuenta que, en el desarrollo del árbol semántico, hay que distinguir de alguna forma las ramas que provienen de operadores clásicos y ramas que representan (partes) del modelo que estamos construyendo. Es decir, en el momento en que en un mundo w_i evaluamos una fórmula como $p \vee q$, estamos teniendo en cuenta que existe un modelo en el que en w_i se evalúa a verdadero p , y otro modelo en el que en w_i se evalúa a verdadero q . La forma más sencilla de gestionar este problema es utilizar el no-determinismo. Para tener una idea de cómo funcionan las reglas, supongamos que tenemos que evaluar la fórmula $\phi = \Diamond(p \vee q) \wedge \Box \neg p \wedge \Box \neg q$, que ya se encuentra en NNF. Esta fórmula no es satisfacible, y por lo tanto, para cualquier elección no determinista que hacemos, nos encontramos con una contradicción en algún mundo. Supongamos que ϕ es satisfacible. Entonces, ϕ se evaluará a verdadero en un mundo w_0 . Esto supone que el mundo w_0 también satisfará a las (sub)fórmulas $\Diamond(p \vee q)$, $\Box \neg p$, y $\Box \neg q$. Ahora, la única fórmula activa que se puede expandir es $\Diamond(p \vee q)$; verificamos que, en este momento, no hay mundos que satisfacen la fórmula $(p \vee q)$, y, por lo tanto, lo creamos; nuestro modelo parcial ahora tiene los mundos w_0 y w_1 , donde w_1 satisface sólo $(p \vee q)$. En este punto, las fórmulas universales se pueden expandir, y en dos pasos verificamos que el mundo w_1 también debe de satisfacer $\neg p$ y $\neg q$. Para desarrollar $p \vee q$, hacemos una elección no determinista: el algoritmo elige una situación en la que w_1 satisface p , llegando a una contradicción, y, con un paso de *backtracking* (vuelta atrás), verifica que también la situación en la que w_1 satisface q supone una contradicción. Por lo tanto la fórmula no es satisfacible. En la Figura 2.6 vemos desarrollado este ejemplo, puesto en forma de árbol para evidenciar las elecciones no deterministas; cada nodo del árbol es un modelo modal parcial.

Un algoritmo genérico para el desarrollo de un árbol semántico por una fórmula en K se puede encontrar en el Algoritmo 2.5. Las reglas modales son:

- Caso existencial. Si estamos en un mundo w_i y tenemos que evaluar la fórmula $\Diamond\phi$, entonces tenemos que averiguar si existe un mundo w_j (posiblemente w_i) tal que en su etiqueta aparece la fórmula ϕ . En este caso, simplemente ponemos el par (w_i, w_j) en la relación R del modelo que estamos construyendo, y ponemos el estado de $\Diamond\phi$ a no activo. Por otra parte, si este mundo no existe, entonces introducimos otro mundo w_j , ponemos en su etiqueta la fórmula ϕ , ponemos el par (w_i, w_j) en la relación R del modelo que estamos construyendo, y cambiamos el estado de $\Diamond\phi$ a no activo.
- Caso universal. Si estamos en un mundo w_i y tenemos que evaluar la fórmula $\Box\phi$, entonces simplemente ponemos la fórmula ϕ en las etiquetas de todos los mundos w_j tales que el par (w_i, w_j) está en la relación R del modelo que estamos construyendo.

La complejidad computacional de K, que se puede demostrar decidible ya que el algoritmo es correcto, completo y termina siempre, es PSPACE completo (o sea, utiliza como mucho un espacio de computación polinomial en la dimensión de la entrada; esta clase de complejidad incluye a NP).

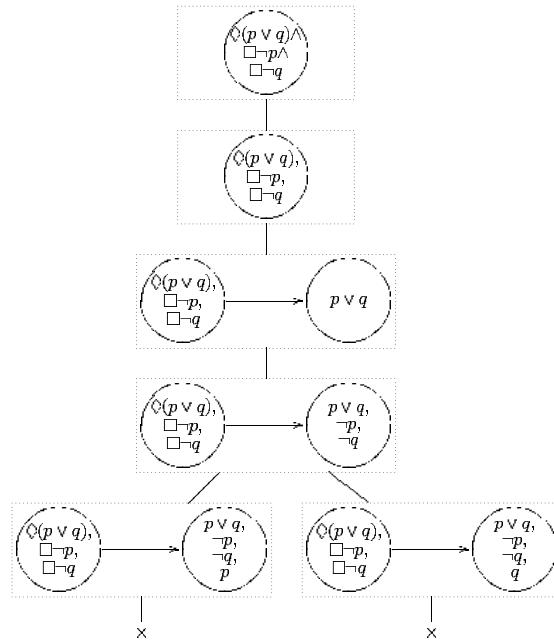


Figura 2.6: Un ejemplo de árbol semántico modal.

Algoritmo 2.5 Un algoritmo para K basado en árboles semánticos.

```

1: Input:  $\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi$ 
2: Transforma el input en  $\phi_1 \wedge \dots \wedge \phi_n \wedge \neg\phi$ 
3: Elimina todo los símbolos de implicación utilizando la correspondiente regla de LP
4: Transforma la fórmula obtenida en NNF
5: bucle
6:   si Hay por lo menos un mundo en el que aparece una fórmula activa  $\phi$  no expandida aun entonces
7:     si  $\phi$  es una fórmula proposicional del tipo  $\psi \vee \tau$  entonces
8:       cambia  $\phi$  a no activa y pon en el mundo actual  $\psi$  o  $\tau$  en estado activo eligiendo de forma no
          determinista*
9:   fin si
10:  si  $\phi$  es una fórmula proposicional de cualquier otro tipo entonces
11:    cambia  $\phi$  a no activa, aplica la regla correspondiente y expande la fórmula en el mundo actual,
        poniendo las fórmulas resultantes en estado activo
12:  fin si
13:  si  $\phi$  es una fórmula modal entonces
14:    cambia  $\phi$  a no activa si y sólo si no hay ninguna (sub)fórmula modal existencial que no haya
        sido expandida aún, aplica la regla correspondiente poniendo las fórmulas resultantes en estado
        activo
15:  fin si
16:  si hay un mundo en el que aparece una contradicción entonces
17:    vuelve atrás a la última elección no determinista*, si hay, o devuelve NO SATISFACIBLE en
        caso contrario
18:  fin si
19: fin si
20: fin bucle
21: devuelve SATISFACIBLE

```

2.5 Aplicaciones: el ejemplo de las lógicas temporales

La lógica modal K tiene numerosas variantes, de las que hemos nombrado sólo algunas. Las lógicas temporales pueden verse como variantes muy peculiares de K (que, además de propiedades sobre la relación, presentan lenguajes con más de un operador modal). Su importancia desde el punto de vista de las aplicaciones es tanta que merecen ser destacadas desde el conjunto de las lógicas modales.

2.5.1 Tipos de lógicas temporales

La principal distinción entre las lógicas temporales es de tipo ontológico. El tiempo, como concepto abstracto, puede verse como compuesto por un conjunto de *puntos* ordenados, o como un conjunto de *intervalos* (pares de puntos) [Bentham, 1989].

En el primer caso, los puntos corresponden a mundos modales, y la evaluación de un símbolo proposicional p corresponde, desde el punto de vista del primer orden, a un predicado unario $p(x)$. El orden lineal no es la única opción; el tiempo puede ser modelado como un orden parcial, por ejemplo un árbol, o un grafo directo. Asimismo, en el caso de un orden lineal, es posible que en algunas aplicaciones resulten útiles algunas propiedades como densidad, discretización, u otras similares. En lógicas temporales basada en puntos, es posible expresar situaciones como

Si el evento A es anterior al evento B, entonces, en el futuro de C habrá una ocurrencia del evento D, después del cual la propiedad E será verificada siempre.

Las estructuras temporales no lineales permiten modelar diferentes futuros o pasados. Con el trabajo de Allen [Allen, 1983; Allen y Ferguson, 1994; Allen y Hayes, 1987, 1985], el estudio en el campo de la lógica temporal ha tomado otra dirección; Allen evidenció que algunos eventos son modelados más precisamente como propiedades *no puntuales* (en cuanto poseen intrínsecamente una duración). Dos propiedades intervalares pueden estar relacionadas entre sí de varias maneras; si consideramos el tiempo como un conjunto de puntos linealmente ordenado, entre dos intervalos (pares de puntos) pueden haber 13 diferentes relaciones. Las lógicas temporales basadas en intervalos se distinguen entre ellas tanto por la ontología del tiempo (lineal, no lineal, discreta, densa,...), como por el tipo y la naturaleza de las modalidades utilizadas.

2.5.2 Lógicas temporales basadas en puntos

Si las propiedades del tiempo que queremos modelar son puntuales, una lógica temporal basada en puntos debe, como mínimo, permitir la expresión de situaciones como *en el futuro del momento actual* o *en el pasado del momento actual*. En la literatura, se suele denotar con F la modalidad unaria sobre puntos para el futuro, y con P su correspondiente en el pasado. Por lo tanto, uno de los lenguajes temporales más sencillo, llamado LTL[F,P] (*Linear Time Logic*), se obtiene con la siguiente gramática abstracta:

$$\phi = p \mid \neg\phi \mid \phi \wedge \psi \mid F\phi \mid P\phi$$

donde las otras connectivas proposicionales se expresan como en el caso de LP, y las relaciones que expresan las modalidades F y P son la una inversa de la otra. Utilizando la misma notación que en el caso de las lógicas modales, un modelo M y un mundo w_i satisfacen la fórmula ϕ si y sólo si:

- $M, w_i \Vdash F\phi$ si y sólo si existe un mundo w_j en el futuro de w_i tal que $M, w_j \Vdash \phi$, es decir, si y sólo si $\exists w_j (w_i < w_j \wedge M, w_j \Vdash \phi)$;
- $M, w_i \Vdash P\phi$ si y sólo si existe un mundo w_j en el pasado de w_i tal que $M, w_j \Vdash \phi$, es decir, si y sólo si $\exists w_j (w_j < w_i \wedge M, w_j \Vdash \phi)$.

Naturalmente, la relación $<$ en este caso es una relación de orden lineal. En un modelo M , todos los símbolos proposicionales interesados toman valor verdadero o falso en cada punto. Propiedades que pueden expresarse en el lenguaje de LTL[F,P] son, por ejemplo, *en el futuro del momento actual valdrá p, y, en su futuro, vale siempre q*, que se formularía como $F(p \wedge \neg F \neg q)$, o *en el futuro del momento actual valdrá p o en el pasado de ese momento, siempre vale q como* $F(p \vee \neg P \neg q)$. La expresión $\neg F \neg$ se suele denotar con G , y la expresión $\neg P \neg$ con H . Cuando el tiempo viene modelado como un conjunto discreto de puntos, se puede añadir un operador modal X que se interpreta como *en el próximo instante, y/o su correspondiente en el pasado* X^{-1} .

Los operadores unarios no son los únicos importantes en lógica temporal. Existen dos operadores binarios, llamados *since* y *until*, y denotados con S y U , que resultan fundamentales en lógica temporal porque permiten añadir un alto poder expresivo al lenguaje. La fórmulas de la lógica llamada LTL (*Linear Time Logic*) se puede obtener con la gramática abstracta:

$$\phi = p \mid \neg\phi \mid \phi \wedge \psi \mid \phi S \psi \mid \phi U \psi$$

y tiene la siguiente semántica:

- $M, w_i \Vdash \phi U \psi$ si y sólo si existe un mundo w_j en el futuro de w_i tal que $M, w_j \Vdash \psi$, y, en todos los mundos w entre w_i y w_j , tenemos que $M, w \Vdash \phi$;
- $M, w_i \Vdash \phi S \psi$ si y sólo si existe un mundo w_j en el pasado de w_i tal que $M, w_j \Vdash \psi$, y, en todos los mundos w entre w_i y w_j , tenemos que $M, w \Vdash \phi$.

El hecho de que LTL sea más expresiva que LTL[F,P] se puede demostrar simplemente observando que la fórmula $F\phi$ corresponde a la fórmula $TU\phi$, y, de manera similar, la fórmula $P\phi$ corresponde a la fórmula $TS\phi$; por otra parte, existen formalismos matemáticos que permiten demostrar que U y S *no pueden representarse* con alguna fórmula de LTL[L,P], con lo cual se demuestra que al pasar de LTL[F,P] a LTL hay un verdadero incremento de expresividad.

Un ejemplo de uso de LTL en el modelado de situaciones reales es el siguiente. Considérese el siguiente párrafo:

El tren utiliza una línea férrea de vía única que se cruza con una carretera. Cuando un tren se está aproximando o cruzando, la luz para los coches debe estar parpadeando. Cuando un tren está cruzando, la barrera debe estar bajada. Si la barrera está subida y la luz apagada, entonces no se aproxima ni cruza ningún tren.

Estas propiedades de un sistema de barrera para la seguridad de un cruce entre una vía de carretera y una línea férrea pueden expresarse de forma sencilla e intuitiva a través de LTL. Por ejemplo *Cuando un tren se está aproximando o cruzando, la luz para los coches debe estar parpadeando*, puede formalizarse con

$$G(p_A \vee p_C \rightarrow p_P)$$

donde p_A (resp., p_C) representa un estado en el que un tren se aproxima (cruza), y p_P representa un estado en el que la luz está parpadeando. De manera similar, la condición *Cuando un tren esta cruzando, la barrera debe estar bajada* se formaliza con

$$G(p_C \rightarrow p_B)$$

La propiedad *Si la barrera está subida y la luz apagada, entonces no se aproxima ni cruza ningún tren* garantiza que no haya errores en el sistema de seguridad, es decir, se ocupa de garantizar que el sistema sea seguro antes de que funcione correctamente en todos los casos:

$$G((\neg p_B \wedge \neg p_P) \rightarrow (\neg p_A \wedge \neg p_C))$$

Como métodos deductivos para LTL (y, por lo tanto, para LTL[F,P]), cabe destacar el método basado en árboles semánticos (véase por ejemplo [Wolper, 1985], que es una adaptación natural del método basado en árboles semánticos para las lógicas modales. Los operadores binarios representan una dificultad en este sentido, y necesitan reglas muy especiales para ser expandidos. En el caso de órdenes lineales discretos, estas reglas están basadas en una caracterización llamada de *punto fijo* y utilizan una observación muy sencilla acerca de la semántica de los operadores binario; un ejemplo es el siguiente: si nos encontramos en un mundo w_i y tenemos que evaluar la fórmula $\phi U \psi$, entonces esto corresponde a evaluar la disyunción $\psi \vee (\phi \wedge X(\phi U \psi))$, es decir, o bien ψ vale ahora, o bien la fórmula ϕ vale ahora y la fórmula $\phi U \psi$ vale en el próximo instante. Ya que sólo un número finito de situaciones diferentes puede ser generado con esta metodología, es posible sintetizar un sistema para controlar las ocurrencias repetidas de fórmulas, y reconocer una situación periódica en tiempo computacional finito. LTL y LTL[F,P] son decidibles independientemente de las características de la clase de órdenes lineales en la que se interpretan (aunque el método de decisión basado en árboles semánticos que hemos visto sólo se aplica en el caso discreto). Otras técnicas, algunas muy complejas, se pueden aplicar en casos más generales.

En IA, merece destacar el trabajo empezado en [Clark y otros, 1986] (la bibliografía en este campo es muy extendida; una referencia reciente es [Visser, 1998]) sobre las lógicas temporales basadas en puntos interpretadas sobre árboles. En este caso, es

possible expresar situaciones en las cuales hay más de un futuro posible. Además de los operadores que permiten, para un cierto futuro, expresar las mismas situaciones que en el caso de LTL, en una lógica interpretada sobre árboles precisamos cuantificadores para los diferentes futuros. Normalmente, se suele denotar con A el operador modal que permite expresar una propiedad de todos los futuros posibles a partir del momento actual, mientras con E el operador que permite elegir un futuro posible. Naturalmente, tenemos que $A\phi = \neg E \neg \phi$. Las propiedades computacionales de las lógicas temporales interpretadas sobre árboles dependen de las limitaciones sintácticas que se pueden añadir a las fórmulas. Un ejemplo es la lógica CTL*, cuyas fórmulas se obtienen a través de la gramática abstracta:

$$\phi = p \mid \neg \phi \mid \phi \wedge \psi \mid \phi S \psi \mid \phi U \psi \mid A\phi \mid E\phi$$

La lógica CTL* es muy expresiva, y su complejidad computacional es muy alta; sin embargo, existen métodos deductivos correctos, completos y que terminan siempre para esta lógica. Por otra parte, cabe destacar que las lógicas temporales basadas en puntos, y en particular las lógicas interpretadas sobre árboles, han sido utilizadas sobre todo para resolver un problema relativamente reciente, llamado *control de modelo*, o *Model Checking*. Este problema no es un problema típico de la lógica, como lo es el comprobar la satisfacibilidad o la validez de una fórmula, pero es un problema de aplicación. La idea es la siguiente: supongamos que un modelo M (en este caso, un árbol donde los nodos tienen una etiqueta, es decir, un conjunto de símbolos proposicionales que se satisfacen en ese punto) representa una situación hipotética, o un sistema, y nos preguntamos si, en ese modelo, es verdad o no que cierto nodo (momento) satisface al menos una fórmula que representa una propiedad interesante. Lo que nos estamos preguntando no es si cierta fórmula ϕ es satisfacible, como hemos hecho hasta ahora, sino si es verdad o no que $M, w \Vdash \phi$, donde M y w son fijos. El problema del control del modelo tiene una complejidad alta para la lógica CTL*; por eso, ha sido estudiada una restricción llamada CTL que, a pesar de su alto poder expresivo, tiene complejidad polinomial para este problema. La lógica CTL es una restricción muy sencilla de CTL*, en la que simplemente se obliga a los cuantificadores sobre caminos a ser utilizados al mismo tiempo que los operadores sobre futuro que provienen de LTL. La versión que sólo permite expresar propiedades del futuro de CTL tiene la siguiente gramática:

$$\phi = p \mid \neg \phi \mid \phi \wedge \psi \mid A(\phi U \psi) \mid E(\phi U \psi)$$

Un sistema inteligente para el razonamiento puede servir también como ayuda durante el desarrollo de otros sistemas. En concreto, el model checking se ha revelado como una tecnología muy útil a la hora de verificar que los requisitos de un proyecto han sido respetados. Antes de escribir físicamente el código de un programa, tenemos el problema de los requisitos; normalmente los requisitos que no han sido formalizados y/o respetados, provocan errores que luego traen un coste adjunto al proyecto. A través de la metodología del model checking, es posible automatizar la tarea de verificación, y disminuir de manera radical el número de errores en el desarrollo de un proyecto. Veamos un ejemplo.

Consideremos un sistema simple de control para una bomba P que extrae agua desde un contenedor A y la lleva a otro contenedor B . Ambos contenedores tienen dos marcadores de nivel, es decir, marcan vacío (V) y lleno (L). Consideramos correcto (OK) el nivel de un contenedor si no está ni vacío ni lleno. Inicialmente, los dos contenedores están vacíos. La bomba P empieza a funcionar en el momento en que el contenedor A marca OK (empezando con vacío), siempre que el contenedor B no marque lleno. P permanece activa mientras A no está vacío y B no está lleno. Por otra parte, P se apaga en el momento en que A se vacía o B se llena. El sistema nunca debería intentar apagar/poner en marcha P cuando P está ya apagada/en marcha.

Lo que deberíamos hacer en primer lugar es formalizar los elementos importantes en la descripción. Los marcadores de cada contenedor pueden por ejemplo ser formalizados como $p_{A,v}$ (A marca vacío), $p_{A,l}$ (A marca lleno), y de la misma manera para B . Asimismo, la variable p_P puede indicar el hecho de que la bomba está en marcha o no. Una vez que sabemos cuáles son las variables proposicionales que nos interesan, podemos, según el valor de verdad que tiene cada una de ellas, identificar el estado del sistema en cada momento. Por ejemplo, si tenemos la tupla $\langle p_{A,v} = V, p_{A,l} = F, p_P = F, p_{B,v} = F, p_{B,l} = F \rangle$, sabemos que el contenedor A está vacío, la bomba está parada, y el contenedor B tiene un nivel óptimo. Está claro que una vez identificado el estado inicial, el sistema puede comportarse de varias maneras, según, por ejemplo, la cantidad de agua que llega al contenedor A . Pero, al tener un conjunto finito de estados posibles, tarde o temprano estos se repetirán. Por lo tanto, la sucesión de estados, que empieza siendo un árbol de posibilidades, puede verse como un grafo dirigido cerrado, en el que se pueden identificar unos cuantos caminos posibles. Ahora, lo que podemos hacer es escribir en un lenguaje como CTL las fórmulas que representan los requisitos de seguridad en los que estamos interesados, y verificar, a través de un sistema de model checking, si estos requerimientos son respetados o no. Por ejemplo, podríamos decir que los marcadores de nivel para un mismo contenedor, no pueden estar a verdadero al mismo tiempo. Esto se podría traducir con

$$\phi = AG(\neg(p_{A,v} \wedge p_{A,l}))$$

donde la conectiva temporal AG indica que queremos que esta propiedad sea respetada en todos los estados y en todos los caminos a los que se puede llegar a partir del estado inicial. O sea, queremos que el modelo M que representa el sistema, y el estado inicial w_0 respeten la propiedad $M, w_0 \Vdash \phi$. Como referencias bibliográficas, aconsejamos [Clarke y otros, 1996].

2.5.3 Lógicas temporales basadas en intervalos

Cuando las propiedades en las que estamos interesados tienen una duración y no es preciso modelarlas como eventos puntuales, entonces podemos utilizar una lógica temporal basada en intervalos. Las relaciones entre intervalos son más complicadas que las relaciones entre puntos; normalmente, en una lógica temporal basada en intervalos,

se utilizan varias modalidades, cada una referida a una diferente relación. Si el modelo del tiempo es lineal, hay trece relaciones diferentes entre dos intervalos (como se ve en la Figura 2.7).

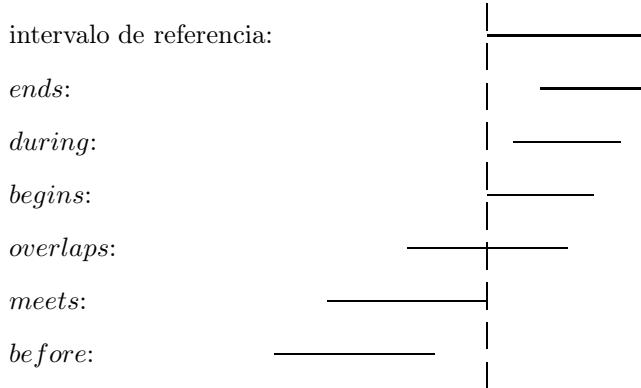


Figura 2.7: Relaciones binarias entre intervalos.

Estas relaciones se suelen denotar con el término inglés *begins* (*inicia*), *ends* (*termina*), y así con el resto.

El trabajo sobre lógicas temporales basada en intervalos es bastante amplio, y las aplicaciones son varias. Una lógica importante es la *Lógica Proposicional de las Relaciones de Allen*, conocida también como HS (de las iniciales de los autores del primer trabajo sobre este tema, Halpern y Shoham [Halpern y Shoham, 1991]). Su sintaxis abstracta es:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle B \rangle \phi \mid \langle E \rangle \phi \mid \langle \bar{B} \rangle \phi \mid \langle \bar{E} \rangle \phi$$

Como se puede ver, son suficientes cuatro modalidades, porque las demás pueden expresarse en términos de éstas. La semántica para una lógica temporal basada en intervalos viene dada en términos de un modelo M y un mundo que se suele denotar como un par ordenado de puntos. Por lo tanto, tenemos que

- $M, [d_0, d_1] \Vdash \langle B \rangle \phi$ si $M, [d_0, d_2] \Vdash \phi$ para algún d_2 tal que $d_0 \leq d_2 < d_1$;
- $M, [d_0, d_1] \Vdash \langle E \rangle \phi$ si $M, [d_2, d_1] \Vdash \phi$ para algún d_2 tal que $d_0 < d_2 \leq d_1$;
- $M, [d_0, d_1] \Vdash \langle \bar{B} \rangle \phi$ si $M, [d_0, d_2] \Vdash \phi$ para algún d_2 tal que $d_1 < d_2$;
- $M, [d_0, d_1] \Vdash \langle \bar{E} \rangle \phi$ si $M, [d_2, d_1] \Vdash \phi$ para algún d_2 tal que $d_2 < d_0$.

Un ejemplo del tipo de situaciones que pueden expresarse en una lógica temporal basada en intervalos es el siguiente.

Si el robot utiliza un procedimiento para cargar su batería, entonces, la próxima vez que utilice el procedimiento de navegación, tendrá la batería cargada.

Una manera de formalizar esta situación es la siguiente:

$$p_C \rightarrow \langle A \rangle \langle A \rangle \langle B \rangle (p_N \wedge p_{BC})$$

donde $\langle A \rangle$ es la modalidad que expresa la relación *meets*, que se puede expresar como $[[EP]]\langle \overline{B} \rangle$, donde $[[EP]]$ es una fórmula capaz de capturar el último punto del intervalo corriente.

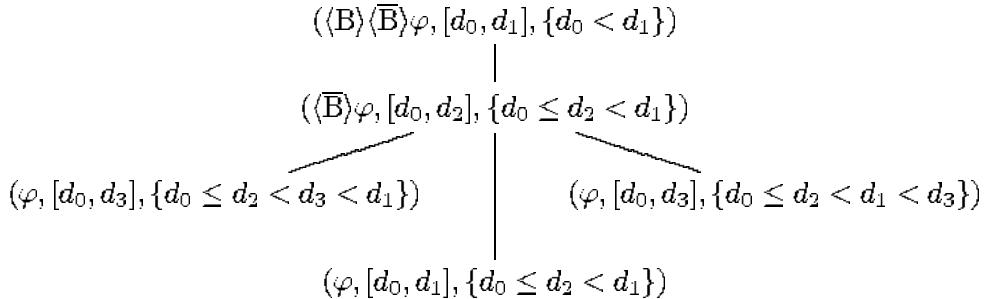


Figura 2.8: Un ejemplo de árbol semántico para una lógica modal de intervalos.

La deducción automática en lógicas temporales basadas en intervalos es mucho más compleja que en el caso de las lógicas temporales basadas en puntos. En la mayoría de los casos, la interpretación en términos de intervalos genera sistemas altamente indecidibles, como en el caso de HS. Existen algunos resultados recientes de lógicas proposicionales basadas en intervalos por las que el problema de la satisfacibilidad es decidible [Montanari y Sciavicco, 2005]. A pesar de su dificultad intrínseca, han sido desarrollados sistemas no terminantes, basados en árboles semánticos, para la deducción automática en las lógicas basadas en intervalos. En [Goranko y otros, 2003; Montanari y Sciavicco, 2005], ha sido presentada una lógica temporal de intervalos que, debido a su alto poder expresivo, permite considerar cualquier otra lógica de intervalos como un fragmento; para esta lógica se ha presentado un sistema basado en árboles semánticos que puede ser adaptado a cualquier lógica temporal de intervalos (a nivel proposicional). La idea básica es muy similar a la metodología para lógicas modales; por otra parte, las relaciones entre mundos no pueden (de forma sencilla) representarse como arcos en un grafo. Por eso, básicamente lo que se hace es, para una fórmula en forma normal ϕ , intentar evaluarla sobre un mundo representado por un par de puntos ordenados:

$$(\phi, [d_0, d_1], \{d_0 < d_1\})$$

y tratar los casos modales según su semántica. Por ejemplo, si $\phi = \langle B \rangle \psi$, tendremos un nuevo elemento en el árbol:

$$(\psi, [d_0, d_2], \{d_0 \leq d_2 < d_1\})$$

El sistema se complica a la hora de tratar fórmulas temporales cuando el conjunto ordenado de puntos tiene muchos elementos. En la Figura 2.8 vemos algunos pasos de un árbol semántico para la fórmula $\langle B \rangle \langle \overline{B} \rangle \phi$.

Se puede decir que no existen en la literatura referencias bibliográficas completas acerca de lógicas temporales de intervalos. El lector interesado, puede consultar [Goranko y otros, 2004].

2.6 Ejercicios resueltos

2.1. Expresar la fórmula $p \rightarrow (q \rightarrow r)$ en términos de \neg y \wedge .

Solución: Como hemos visto, todos los operadores se pueden expresar en términos de \neg y \wedge . La fórmula considerada, puede verse, en forma abstracta, como $p \rightarrow \phi$, que, con las reglas que conocemos, es equivalente a la fórmula $\neg p \vee \phi$. Ahora, eliminamos la disyunción, obteniendo $\neg(\neg p \wedge \neg \phi)$, que es equivalente a $\neg(p \wedge \neg \phi)$. Pero la fórmula ϕ es $(q \rightarrow r)$, es decir, $\neg q \vee r$, lo que significa que, juntando todo, obtenemos $\neg(p \wedge \neg(\neg q \vee r))$, o sea, $\neg(p \wedge q \wedge \neg r)$.

■

2.2. Verificar con el algoritmo del árbol semántico que la fórmula $(p \wedge q) \rightarrow r \rightarrow (p \rightarrow (q \rightarrow r))$ es válida.

Solución: En el lenguaje de LP, una fórmula es válida si y sólo si, como hemos visto, su negación es insatisfacible. Una forma de negar la fórmula que estamos considerando es $(p \wedge q) \rightarrow r \wedge \neg(p \rightarrow (q \rightarrow r))$. En la Figura 2.9 vemos el desarrollo de un árbol semántico para esta fórmula.

Como todas las ramas están cerradas, la fórmula de salida es insatisfacible, lo que implica que la fórmula considerada en el problema es válida.

■

2.3. Verificar con el algoritmo basado en la regla de resolución que la fórmula $(p \wedge q) \rightarrow r \rightarrow (p \rightarrow (q \rightarrow r))$ es válida.

Solución: Del mismo modo que en el problema anterior, tenemos que averiguar si, a través de la resolución, podemos encontrar la cláusula vacía saliendo de la fórmula $\neg \phi$. La negación de la fórmula que estamos considerando es $(p \wedge q) \rightarrow r \wedge \neg(p \rightarrow (q \rightarrow r))$. Aplicamos las reglas necesarias para convertir esta fórmula en forma clausal, obteniendo $(\neg(p \wedge q) \vee r) \wedge (\neg p \vee (\neg q \vee r))$, o sea, $(\neg p \vee \neg q \vee r) \wedge \neg(\neg p \vee \neg q \vee r)$. Esta fórmula da lugar a cuatro cláusulas, es decir: $\Gamma_1 = \{\neg p, \neg q, r\}$, $\Gamma_2 = \{p\}$, $\Gamma_3 = \{q\}$, $\Gamma_4 = \{\neg r\}$. En la Figura 2.10 vemos los pasos de resolución necesarios para llegar a

la cláusula vacía. El hecho de alcanzar una cláusula vacía, demuestra que la fórmula de salida es insatisfacible, y que, por lo tanto, su negación es una fórmula válida.

■

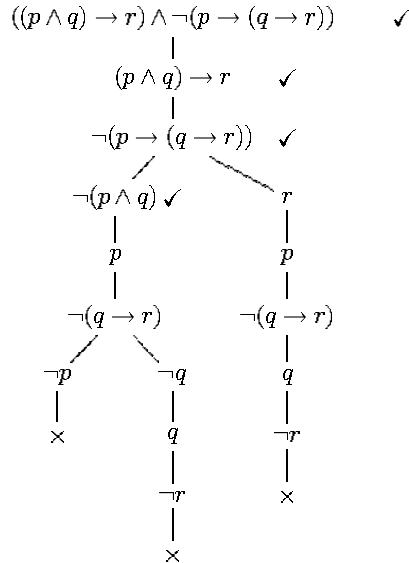


Figura 2.9: Árbol semántico para la fórmula $(p \wedge q) \rightarrow r \wedge \neg(p \rightarrow (q \rightarrow r))$.

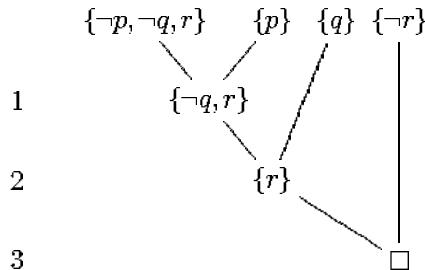


Figura 2.10: Resolución para la fórmula $(p \wedge q) \rightarrow r \wedge \neg(p \rightarrow (q \rightarrow r))$.

2.4. Formalizar la siguiente situación: “Es necesario recoger todos los paquetes que están en el suelo (punto A). Sólo es posible recoger un paquete si el brazo mecánico está vacío (no está sujetando algún paquete). Si el brazo está sujetando un paquete, entonces puede dejarlo en el punto B.” ¿Qué lenguaje es más adecuado para formalizar esta situación?

Solución. Es evidente que la situación descrita en el párrafo presenta un dominio de objetos que en principio no podemos cuantificar. Por lo tanto, el lenguaje de LPO es el más adecuado para el caso. La frase “es necesario recoger todos los paquetes” indica un objetivo, o tarea, que podemos formalizar como un predicado $p_T(t)$. La variable t denota el momento temporal en el que hacemos la afirmación. Para los momentos temporales, podemos utilizar un predicado binario $<$. La realización de dicha tarea depende de la posición de todos los objetos, denotados con variables, en el punto A ($p_A(x, t)$). Por lo tanto tenemos:

$$\forall x \forall t p_A(x, t) \leftrightarrow p_T(t).$$

La posibilidad de recoger un objeto y llevarlo desde el punto B al punto A depende del estado del brazo ($p_{BR}(x, t)$); es conveniente ahorrar símbolos, y utilizar $p_{BR}(x, t)$ para decir tanto que x está siendo sujetado por el brazo (no está vacío), o que el brazo se encuentra vacío ($\neg \exists x p_{BR}(x, t)$). Por lo tanto:

$$\forall x \forall t (p_B(x, t) \wedge \neg \exists x p_{BR}(x, t)) \rightarrow \exists t' (t < t' \wedge p_{BR}(x, t')).$$

Asimismo, en cada momento, si el brazo sujeta algo tiene que dejarlo en el punto A :

$$\forall x \forall t (p_{BR}(x, t)) \rightarrow \exists t' (t < t' \wedge p_A(x, t') \wedge \neg p_{BR}(x, t')).$$

■

2.5. Averiguar con la metodología basada en árboles semánticos si la fórmula $\phi = \forall x \exists y (p(x, y)) \wedge \exists x (r(x)) \wedge \forall x \forall y (p(x, y) \rightarrow \neg r(x))$ es satisfacible o no.

Solución. Al ser una fórmula de primer orden, podríamos encontrarnos con un árbol semántico que nunca termina. Por otra parte, si el árbol tuviese todas las ramas cerradas podríamos concluir que la fórmula no es satisfacible. Si desarrollamos directamente la fórmula de salida, el árbol que resulta es como el de la Figura 2.11. Al tener todas las ramas cerradas, podemos concluir que la fórmula no es satisfacible. Es importante destacar que hemos introducido dos constantes nuevas (que no aparecían en el lenguaje) para expandir operadores existenciales.

■

2.6. Averiguar con la metodología basada en la regla de resolución si la fórmula $\phi = \exists x (\neg p(x) \rightarrow p(f(x))) \wedge \forall x (q(x)) \rightarrow \exists x (p(x) \wedge q(x))$ es válida o no.

Solución. Como en el problema anterior, al ser una fórmula de primer orden lo único que podemos hacer es suponer que ϕ sea válida, y aplicar la resolución a la fórmula $\neg \phi$, para averiguar si encontramos la cláusula vacía. La negación de ϕ puede escribirse así: $\exists x (\neg p(x) \rightarrow p(f(x))) \wedge \forall x (q(x)) \wedge \neg \exists x (p(x) \wedge q(x))$. Como estrategia para disminuir el número de pasos a utilizar, observamos que podemos tratar las tres fórmulas de la conjunción como tres cláusulas diferentes. La primera fórmula nos devuelve:

1. $\exists x(\neg p(x) \rightarrow p(f(x)))$ (fórmula de salida);
2. $\exists x(\neg\neg p(x) \vee p(f(x)))$ (eliminación de \rightarrow);
3. $\exists x(p(x) \vee p(f(x)))$ (eliminación de $\neg\neg$);
4. $(p(a) \vee p(f(a)))$ (forma de Skolem);
5. $\Gamma_1 = \{p(a), p(f(a))\}$ (forma causal).

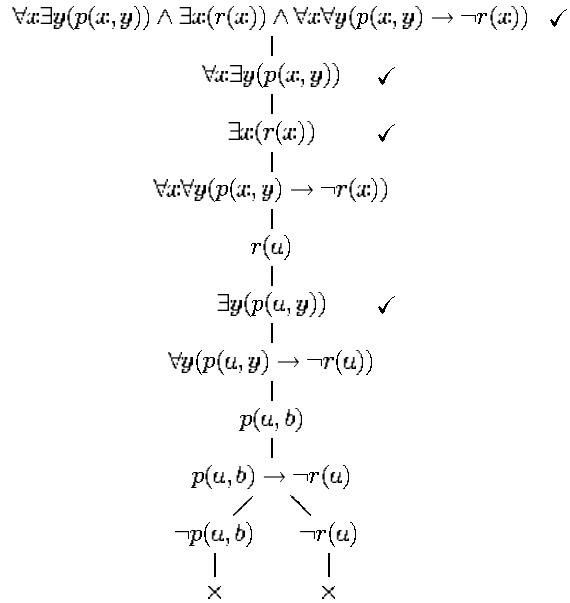


Figura 2.11: Árbol semántico para la fórmula $\forall x\exists y(p(x,y)) \wedge \exists x(r(x)) \wedge \forall x\forall y(p(x,y) \rightarrow \neg r(x))$.

La segunda fórmula:

1. $\forall x(q(x))$ (fórmula de salida);
2. $q(x)$ (forma de Skolem);
3. $\Gamma_2 = \{q(x)\}$ (forma clausal).

Finalmente, la tercera fórmula:

1. $\neg\exists x(p(x) \wedge q(x))$ (fórmula de salida);
2. $\forall x\neg(p(x) \wedge q(x))$ (interdefinición de \exists y \forall);
3. $\forall x(\neg p(x) \vee \neg q(x))$ (regla de De Morgan);
4. $(\neg p(x) \vee \neg q(x))$ (forma de Skolem);
5. $\Gamma_3 = \{\neg p(x), \neg q(x)\}$ (forma clausal).

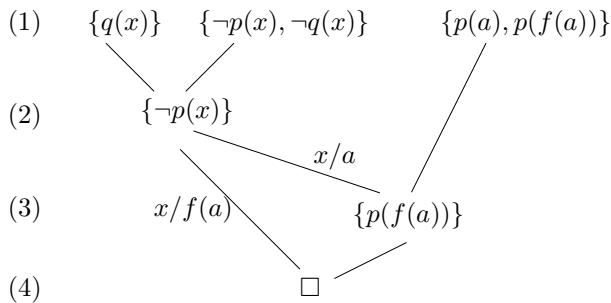


Figura 2.12: Desarrollo del ejercicio de resolución de primer orden.

En la Figura 2.12 vemos el desarrollo de la resolución para esta fórmula, que nos devuelve la cláusula vacía, demostrando que la fórmula $\neg\phi$ no es satisfacible, y, por lo tanto, ϕ es válida.

2.7. Consideremos un modelo modal $M = (W, R, V)$ para la lógica K, tal que la fórmula $\phi = \Diamond\Diamond p \rightarrow \Diamond p$ sea válida en M , es decir, tal que en todos los mundos $w \in W$ la fórmula ϕ se satisface en w cualquiera que sea la evaluación V . ¿Qué propiedad algebraica tiene la relación R ?

Solución. Algunas (no todas) de las propiedades algebraicas de la relación en la que está basado un modelo modal pueden expresarse a través de una fórmula del lenguaje de LM. Supongamos que en M , para cualquier mundo $w \in W$ y cualquier evaluación V , tenemos que $M, w \Vdash \phi$. Es sencillo observar que, de todas las posibles evaluaciones, las únicas que pueden interesarnos son las que tienen que ver con el único símbolo proposicional p . Entonces, consideremos un mundo w' tal que, a partir de w , podemos alcanzar w' con, al menos, dos pasos a través de R (cualquier otro mundo no permite evaluar a verdadera el antecedente de la implicación, con lo cual ϕ se realizaría siempre). Supongamos que $p \in V(w')$ (p se realiza en w'); siendo ϕ evaluada a verdadero en w' , esto implica que, sea cual sea la evaluación V , tiene que haber un mundo, alcanzable en un solo paso de R , en el que se evalúa p a verdadero. El único mundo con estas características es w' , con lo cual tenemos que R debe de permitir alcanzar w' a partir de w en un solo paso. Ahora, estas observaciones no dependen del mundo de salida, con lo cual el mismo razonamiento se puede repetir para cualquier mundo; entonces, la propiedad que tenemos es *todo mundo alcanzable en dos pasos, también se puede alcanzar en un solo paso*, es decir, R es *transitiva*.

■

2.7 Ejercicios propuestos

2.1. Utilizando el principio de la mutua exclusión (no admitimos mundos en que p y $\neg p$ sean ciertas al mismo tiempo), resolver este famoso juego de lógica:

Un matemático y un físico están sentados en una mesa. ‘Yo soy matemático’ dice el hombre con el pelo negro. ‘Yo soy físico’ dice el hombre con el pelo gris. Sabiendo que al menos uno de ellos miente, ¿cuál de ellos dice la verdad? ¿De qué color es el pelo del físico?

2.2. ¿Cuáles de las siguientes fórmulas están bien formadas?

- $p \vee$;
- $((p \wedge q))$;
- $(q \rightarrow (p \vee r \wedge \neg q \neg))$

2.3. Diseñar un algoritmo capaz de reconocer si una fórmula en el lenguaje de LP está bien formada o no. Evidenciar los aspectos relativos a la representación de las fórmulas: ¿cuál es la manera más eficiente para representar una fórmula?

2.4. Expresar la fórmula $p \leftrightarrow q$ en términos de las conectivas \neg y \wedge .

2.5. Completar el ejemplo de la sección 2.2 .

2.6. Un extraterrestre llega a la Tierra, y, hablando de su mundo con algunos científicos, intenta convencerlos de que toda nuestra lógica proposicional, desde el principio de los tiempos, está equivocada, ya que ellos conocen un operador ternario llamado $*(p, q, r)$ que nosotros no somos capaces de expresar, y que tiene la siguiente tabla de verdad:

p	q	r	$*(p, q, r)$
V	V	V	V
V	V	F	V
V	F	V	F
V	F	F	F
F	V	V	V
F	V	F	V
F	F	V	V
F	F	F	F

¿Cómo demuestran los científicos al extraterrestre que se está equivocando?

2.7. Poner las siguientes fórmulas en forma clausal:

- $p \wedge (q \rightarrow (r \vee q)) \vee (p \rightarrow q)$;
- $(p \rightarrow q) \wedge (q \rightarrow r) \wedge \neg r \wedge p$;

- $\neg(p \rightarrow (p \leftrightarrow q) \leftrightarrow q);$
- $\neg((p \vee \neg p \rightarrow t) \rightarrow (q \rightarrow t)).$

2.8. Aplicar el método de resolución a las fórmulas anteriores.

2.9. Aplicar el método del árbol semántico a las siguientes fórmulas:

- $\neg(p \rightarrow (q \rightarrow p \wedge q));$
- $((r \vee q) \wedge (p \rightarrow q)) \rightarrow (\neg(p \rightarrow \neg q));$
- $((p \rightarrow q \wedge r) \wedge (q \wedge r \rightarrow s) \wedge (s \rightarrow t) \wedge p) \rightarrow t.$

2.10. Utilizando el método del árbol semántico, resolver este famoso juego de lógica:

Nos encontramos en un cruce del que salen dos carreteras, una que termina en un pueblo donde todos sus vecinos dicen siempre la verdad, y la otra en un pueblo donde todos sus vecinos siempre mienten. Encontramos, en el medio del cruce, una persona, que sabemos que ha llegado por la carretera de la derecha, pero no sabemos si el vecino del pueblo de los mentirosos o de los sinceros, ni sabemos cuáles de las dos carreteras lleva a cada uno de los pueblos. ¿Qué pregunta deberíamos hacerle para averiguar adónde llevan las dos carreteras, suponiendo que disponemos solamente de una pregunta?

2.11. ¿Cuáles de las siguientes fórmulas están bien formadas?

- $\forall x(p(x) \vee);$
- $\forall x(p(y) \rightarrow r);$
- $\forall x(r(x, y, z) \rightarrow \exists y q(z)) \vee \exists x \exists y(p(x, y) \wedge \exists z)$

2.12. Utilizar el método del árbol semántico para averiguar si las siguientes fórmulas son satisfacibles:

- $\forall x, y(p(x) \rightarrow \neg q(x, y)) \wedge \exists x, y(p(x, y)) \wedge \forall z(p(z));$
- $p(a) \wedge \forall x(p(x) \rightarrow r(c, x)) \wedge r(a, b).$

2.13. Considérese un modelo M con un dominio compuesto por tres elementos a, b, c , un predicado unario p tal que $p = \{a\}$, y un predicado binario q tal que $q = \{(a, b), (b, a), (a, c)\}$. ¿Cuáles de las siguientes fórmulas son satisfacibles en M ?:

- $\forall x(p(x) \leftrightarrow p(x));$
- $\forall x(p(x) \rightarrow \exists y(q(x, y)));$
- $p(b) \rightarrow \exists x \forall y(q(x, y));$
- $\exists x \forall y(q(x, y)).$

2.14. *Implementar un algoritmo capaz de averiguar si dos términos s y t son unificables.

2.15. *Diseñar un algoritmo capaz de poner en forma clausal una fórmula de primer orden que se encuentre en forma prenexa.

2.16. Utilizar el método de resolución para demostrar que las siguientes fórmulas son válidas:

- $\exists x(p(x) \rightarrow p(f(x)))$;
- $\exists x(p(f(x)) \rightarrow p(x))$;
- $\forall x\forall y(p(x, y) \rightarrow \neg p(y, x)) \rightarrow \neg\exists x p(x, x)$;
- $(\exists x(\neg p(x) \rightarrow p(f(x))) \wedge \forall x q(x)) \rightarrow \exists x(p(x) \wedge q(x))$.

2.17. Completar el ejemplo de la sección 2.3.1.

2.18. Utilizando el método de resolución, resolver este famoso juego de lógica:

Tenemos una estantería con seis libros de géneros diferentes, y disponemos de las siguientes informaciones: el ensayo se encuentra a la derecha del libro de cuentos, pero no está a su lado; la novela está tanto al lado del policíaco como al lado del libro de aventuras; el thriller se encuentra en un extremo y al lado del ensayo; entre el libro de cuentos y el policíaco hay exactamente dos libros. ¿Es verdad que el libro de cuentos se encuentra a la izquierda y al lado del libro de aventuras?

2.19. *Consideremos un modelo modal $M = (W, R)$ para la lógica K, tal que la fórmula $\phi = p \rightarrow \Diamond p$ sea válida en M , es decir, tal que en todos los mundos w en W la fórmula ϕ se satisface en w sea cual sea la evaluación V . ¿Qué propiedad algebraica tiene la relación R ?

2.20. Formalizar en el lenguaje LM el siguiente párrafo: “El robot empieza su tarea en el estado A. Si es posible que en algún momento el robot se encuentre en el estado B, y que en algún momento el robot se encuentre en el estado C, entonces, teniendo en cuenta que los estados A,B,C no son compatibles, es necesario que si el robot se encuentra en el estado B, también, en algún otro momento, se encuentre en el estado C.”

2.21. El Axioma modal llamado K permite distinguir las lógicas modales *normales* de las que no lo son. Este Axioma tiene varias formas, unas de las cuales es $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$. Verificar utilizando el algoritmo basado en árboles semánticos para LM que K es una fórmula válida.

2.22. Demostrar semánticamente que la fórmula $Pp \wedge (Pp \rightarrow Gq) \wedge F\neg q$ no es satisfacible. Una forma de razonamiento, en este caso, puede utilizar la traducción de la fórmula de primer orden.

2.23. *Adaptar el algoritmo de deducción basado en árboles semánticos dado para el lenguaje LM al lenguaje LTL[F,P].

2.24. *Formalizar en el lenguaje de LTL las características de seguridad y de funcionamiento básicas de un ascensor dotado de una puerta automática, excluyendo las funcionalidades de memorización de llamadas.

Referencias

- ALLEN, J.F.: «Maintaining Knowledge about Temporal Intervals». *Communications of the ACM*, 1983, **26(11)**, pp. 832–843.
- ALLEN, J.F. y FERGUSON, G.: «Actions and Events in Interval Temporal Logic». *Journal of Logic and Computation*, 1994, **4(5)**, pp. 531–579.
- ALLEN, J.F. y HAYES, P. J.: «Short Time Periods». En: *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pp. 981–983, 1987.
- ALLEN, J.F. y HAYES, P.J.: «A Common-sense Theory of Time». En: *Proc. of the 9th International Joint Conference on Artificial Intelligence*, pp. 528–531. Morgan Kaufmann, 1985.
- BENTHEM, J. VAN: *The logic of time: a model-theoretic investigation into the varieties of temporal ontology and temporal discourse*. volumen 156 de *Synthese Library*, Reidel. Springer-Verlag, 1989.
- BLACKBURN, P.; DE RIJKE, M. y VENEMA, Y.: *Modal Logics*. Cambridge University Press, 2002.
- BOOLOS, G.S.; BURGESS, J.P. y JEFFREY, R.C.: *Computability and Logic*. Cambridge University Press, 2002.
- CABALLERO, A.F.; MANZANO, M.G.; ALONSO, E. y TOMÉ, S.M. (Eds.): *Actas del Campus Multidisciplinar en Percepción e Inteligencia, VOL I-II*. Albacete, España, 2006.
- CLARK, E.M.; EMERSON, E.A. y SISTLA, A.P.: «Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications». *ACM Transaction on Programming Languages and Systems*, 1986, **8(2)**, pp. 244–263.
- CLARKE, E.M.; O.GRUMBERG y PELED, D.A.: *Model Checking*. The MIT Press, 1996.
- D'AGOSTINO, M.; GABBAY, D.; HÄHNLE, R. y POSEGGA, J. (Eds.): *Handbook of Tableau Methods*. Kluwer Academic Press, 1999.
- E.BORGER, E; E., GRADEL y Y., GUREVICH: *The Classical Decision Problem*. Springer-Verlag, 1988.
- FRICK, M. y GROHE, M.: «The complexity of first-order and monadic second-order logic revisited». En: *Proc. of the 2002 Conference on Logics in Computer Science*, pp. 215–224, 2002.
- GABBAY, D.M.; HOGGER, C.J. y ROBINSON, J.A. (Eds.): *Handbook of Logic in Artificial Intelligence and Logic Programming*. volumen 3. Oxford Science Publication, 1994.

- GORANKO, V.; MONTANARI, A. y SCIavicco, G.: «Propositional interval neighborhood temporal logics». *Journal of Universal Computer Science*, 2003, **9(9)**, pp. 1137–1167.
- GORANKO, V.; MONTANARI, A. y SCIavicco, G.: «A Road Map on Interval Logics and Duration Calculi». *Journal of Applied Non Classical Logics*, 2004, **(1/2)**, pp. 11–56.
- HALPERN, J.Y. y SHOHAM, Y.: «A Propositional Modal Logic of Time Intervals». *Journal of the ACM*, 1991, **38(4)**, pp. 935–962.
- IMMERMAN, N. y KOZEN, D.: «Definability with Bounded Number of Bound Variables». En: *Logic in Computer Science*, pp. 236–244, 1987.
- KOWALSKI, R.: *Logic for problem solving*. Elsevier Science, 1979.
- MCCARTHY, J.: «Circumscription: A Form of Non-Monotonic Reasoning». *Artificial Intelligence*, 1987.
- MONTANARI, A. y SCIavicco, G.: «A Decidability Proof for Propositional Neighborhood Logic», 2005. Contributed Talk, Trends in Logics III Conference, Warsaw - Ruciane Nida (Poland).
- MUCHNICK, S.: *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.
- PADAWITZ, P.: *Computing in Horn clause theories*. Springer-Verlag, New York, USA, 1988.
- PRIEST, G.: *An Introduction to Non-Classical Logic*. Cambridge University Press, 2001.
- REITER, R.: «On closed world data bases». En: *Logic and data bases*, Plenum Press, 1978.
- REITER, R.: «A Logic for Default Reasoning». *Artificial Intelligence*, 1980, **13**.
- SCHECHTER, E.: *Classical & Nonclassical Logics*. Princeton University Press, 2005.
- SCOTT, D.: «A Decision Methos for Validity of Sentences in two Variables». *Journal of Symbolic Logic*, 1962, **27**, pp. 377–546.
- VARDI, M.Y.: «Why Is Modal Logic So Robustly Decidable?» *Informe técnico TR97-274*, 1997.
- VISSEER, W.C.: *Efficient Model Checking using Games and Automata*. Tesis doctoral, 1998.
- WOLPER, P.: «The Tableau Method for Temporal Logic: An Overview». *Logique et Analyse*, 1985, **28**, pp. 119–136.

Capítulo 3

Sistemas basados en reglas

María Jesús Taboada Iglesias¹ y Asunción Gómez Pérez²

Universidad de Santiago de Compostela¹ y Universidad Politécnica de Madrid²

3.1 Introducción

Los formalismos de representación del conocimiento proporcionan las herramientas necesarias para codificar la realidad en un computador. En este tema, nos centraremos en el uso de las reglas como un esquema de representación del conocimiento. Los Sistemas Basados en Reglas (SBR) constituyen un campo de estudio importante dentro de la IA, porque permiten capturar la experiencia humana en la resolución de problemas, con el fin de alcanzar decisiones consistentes y repetibles. En ellos la representación del conocimiento se identifica claramente con las heurísticas o formas de proceder de los expertos. Son especialmente interesantes en aquellos dominios en los que escasean los expertos (medicina, ingeniería, etc.), ya que proporcionan un medio eficaz para difundir ampliamente razonamientos escasos y específicos. Además, en nuestra vida diaria nos encontramos con muchos escenarios que están gobernados por reglas deterministas. Ejemplos de dichas situaciones son el control del tráfico de coches, de la seguridad de personas, de las transacciones bancarias, etc.

En los SBR, las estructuras de representación son declarativas, por lo que existe una separación explícita entre el conocimiento del dominio y los mecanismos de deducción utilizados. Por una parte, el conocimiento del dominio puede ser factual o heurístico. El primero es el conocimiento del dominio aceptado, y generalmente consensuado, por todos los expertos en un campo de aplicación específico. El segundo, el conocimiento heurístico, es mucho menos riguroso y está basado en la experiencia propia de cada experto o grupo de expertos. Se dice que es el conocimiento de la buena práctica y el buen juicio. Por otra parte, los mecanismos de deducción utilizados controlan las etapas que se deben llevar a cabo para resolver un problema. Dichos mecanismos se conocen como *motores de inferencia* y son los que aplican el conocimiento para crear una línea de razonamiento. Los SBR son una aplicación de los sistemas de deducción en lógica proposicional o de primer orden, restringidos a cláusulas de Horn en primera instancia (véase el capítulo 2). Utilizan las reglas de

inferencia de la lógica (razonamiento deductivo) para llegar a obtener conclusiones lógicas, interpretando dichas reglas de la siguiente manera: a) como reglas de la forma condición-acción en un control con encadenamiento hacia delante, o b) como conjuntos de implicaciones lógicas a partir de las cuales se obtienen las deducciones, en un control con encadenamiento hacia atrás.

Comenzaremos el tema viendo los elementos básicos de un SBR: el motor de inferencias (o intérprete de reglas), la base de conocimiento y la base de hechos (o memoria de trabajo). A continuación, en la sección 3.3 estudiaremos la direccionalidad de la inferencia en los SBR, abarcando las dos técnicas básicas de encadenamiento: hacia delante y hacia atrás, y el problema de la definición de reglas en una forma tal que puedan utilizarse reversiblemente. La forma de selección de las reglas que se aplican en una etapa dada de la inferencia, a partir de un conjunto completo de reglas, necesita algún tipo de equiparación entre el estado actual de la base de hechos y las condiciones de las reglas. En la sección 3.4 veremos diversas técnicas de equiparación y la sección 3.5 se dedicará al estudio de diversas técnicas de resolución de conflictos. El paradigma de las reglas como método computacional da lugar a una forma de programar diferente de la convencional. Finalizaremos el tema comparando la representación de reglas frente a la programación procedural y señalando las ventajas e inconvenientes de los SBR con respecto a otros formalismos, así como los dominios adecuados para los cuales las reglas pueden ser un esquema útil de representación del conocimiento.

3.2 Componentes básicos de los SBR

Un SBR (véase la Figura 3.1) es un sistema que tiene una base de conocimiento (BC) con reglas y algún mecanismo de inferencias (MI) que selecciona las reglas que se pueden aplicar y las ejecuta, con el objetivo de obtener alguna conclusión (es decir, de realizar algún procesado o interpretación del conocimiento). El sistema contiene también una base de hechos (BH) o memoria de trabajo que acumula un conjunto de hechos establecidos, que se usan para determinar qué reglas puede aplicar el mecanismo de inferencias. Este conjunto de hechos puede existir como parte de las condiciones iniciales del sistema, se pueden añadir durante el proceso de inferencias del sistema, el usuario puede introducirlos durante el uso del sistema o pueden proceder de sistemas externos, tales como sensores o bases de datos.

Además, para que un SBR llegue a ser realmente útil ha de estar dotado de facilidades de entrada/salida sofisticadas, que faciliten el proceso de consulta, y el desarrollo y refinamiento del sistema. Dichas facilidades se conocen como interfaces de usuario.

3.2.1 Base de Hechos

La base de hechos (BH) o memoria de trabajo contiene toda la información actual del problema o tarea a resolver, es decir, los datos y hechos establecidos hasta el momento en la tarea, las metas a alcanzar y las hipótesis avanzadas en el curso de una solución. En la BH es donde se guardan todos los cambios de información que se

producen en el sistema durante cada proceso de inferencias. Es decir, en el sistema no se guarda más información que la que existe en la BH, que representa el estado actual del problema en curso de solución.

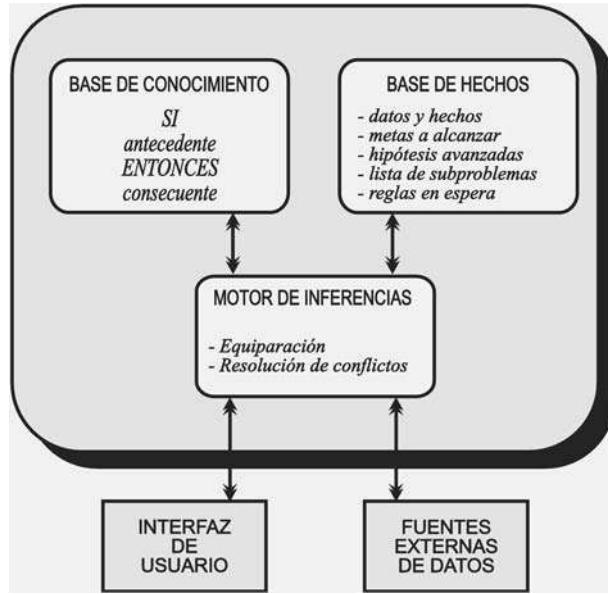


Figura 3.1: Componentes básicos de un SBR.

3.2.2 Base de Conocimiento

Una regla consta de dos partes: una izquierda, denominada condición o antecedente, y una derecha, llamada consecuente o de acción. Así, definiremos una regla como un par condición-acción. El antecedente contiene una lista de cláusulas a verificar y el consecuente una lista de acciones a ejecutar. Habitualmente las reglas se representan por medio de una flecha que conecta la condición con la acción:

$$\text{Condición} \rightarrow \text{Acción}$$

Algunos ejemplos de reglas en distintos dominios de aplicación son los que se indican a continuación.

Paciente menor de 10 años con manchas rojas y fiebre → Paciente con varicela
Coche no arranca → Revisar batería
El dólar baja → Comprar dólares
Envío urgente de un paquete → Incremento de tarifa base en 10 euros

Las reglas operan sobre el espacio de trabajo de la BH. La condición expresa algún tipo de test sobre el contenido de la BH, que se puede verificar o no. Si se verifica el

test de una regla, se puede ejecutar su parte de acción. La ejecución de una acción puede cambiar el contenido de la BH. En el ejemplo de envío urgente de un paquete, la ejecución de la regla incrementa en 10 euros el coste base (que estará almacenado en la BH). En una forma más general, la parte de condición de una regla puede consistir en un conjunto de varios test elementales. En el ejemplo del paciente con varicela, para que la regla se llegue a ejecutar se deben verificar simultáneamente tres condiciones: el paciente debe ser menor de 10 años, debe presentar manchas rojas y registrar fiebre. Además, la BH contiene:

- El conjunto de hechos que han podido ser deducidos por el sistema, si el mecanismo de control ejecuta un encadenamiento hacia adelante. Por ejemplo, ante una petición de envío urgente de un paquete, se deduce que hay que incrementar en 10 euros el precio base.
- Un conjunto de metas que deben alcanzarse e hipótesis avanzadas en el curso de una solución, si el encadenamiento es hacia atrás. Por ejemplo, en una consulta médica, una meta puede ser determinar si un paciente padece una cierta enfermedad, como la varicela. Para ello, será necesario chequear si se verifican todas las cláusulas del antecedente de la regla (o reglas) que permiten deducir la presencia de dicha enfermedad. En nuestro ejemplo, será necesario preguntar la edad del paciente, explorarle para ver si presenta manchas rojas y medir la fiebre.

Por otra parte, la BH es el foco de atención de las reglas: éstas operan sobre este espacio de trabajo, de forma que la BH es el único punto de unión entre ellas. Una diferencia importante entre la BH y la BC es que la primera contiene información puntual sobre la tarea a realizar, como memoria de trabajo que es, mientras que la BC almacena segmentos de conocimiento relacional entre datos y conceptos. Por ejemplo, la BH contiene datos como *el paciente tiene 36 años, pesa 75 Kg., mide 1,73, sufrió infarto agudo de miocardio*, mientras que la BC contiene conocimiento como *si la tensión arterial sistólica es mayor que 160 mmHg y menor que 179 mmHg y la tensión arterial diastólica es mayor que 90 y menor que 99, entonces el paciente padece una hipertensión arterial leve*.

Las reglas son similares a los condicionales If-Then de la mayoría de los lenguajes de programación. Un condicional If-Then puede verse como una operación de implicación (desde el punto de vista booleano) o como un procedimiento condición-acción (desde el punto de vista de una orden a ejecutar). Una diferencia importante entre los SBR y la programación basada en procedimientos es la forma de seleccionar una condición entre varias que se satisfacen simultáneamente. En la aproximación basada en procedimientos, se selecciona la siguiente condición en una secuencia de condicionales (cuyo orden ha sido previamente programado), mientras que en un SBR el mecanismo de control tiene en cuenta diferentes factores para tomar la decisión y ésta se realiza durante la ejecución. El proceso en el que se toma la decisión se llama *resolución de conflictos*, tal y como veremos a continuación.

3.2.3 Motor de inferencias

El motor de inferencias (MI), la estrategia de control o el intérprete de reglas es el mecanismo que sirve para examinar la BH y decidir qué reglas se deben disparar. El esquema general de funcionamiento se puede ver en el Algoritmo 3.1:

Algoritmo 3.1 Algoritmo genérico de funcionamiento del motor de inferencias.

```

1: BH = HechosIniciales;
2: mientras NoVerificaCondiciónFinalización(BH) o NoseEjecutaAccióndeParada hacer
3:   ConjuntoConflictos = Equiparar(BC,BH);
4:   R=Resolver(ConjuntoConflictos);
5:   NuevosHechos = Aplicar(R,BH);
6:   Actualizar(BH,NuevosHechos);
7: fin mientras
```

Como puede verse, el Algoritmo 3.1 ejecuta un bucle mientras no se verifican dos condiciones (etapa 2 del algoritmo): una condición de finalización y una acción de parada. La condición de finalización, en general, indica el hecho *meta* que tiene que alcanzarse. Esta meta se alcanzará cuando esté contenida como hecho en la BH. La ejecución de alguna acción de parada, en general, se produce cuando el procedimiento no tiene éxito en la búsqueda de un conjunto de reglas que permitan alcanzar dicha meta.

La búsqueda del conjunto de reglas que se pueden aplicar a la BH (etapa 3) se denomina *equiparación* y consiste en seleccionar las reglas cuyas condiciones o acciones sean compatibles con los datos almacenados en ese ciclo en la BH. En la sección 3.4 del tema veremos diversas técnicas de equiparación. El conjunto de reglas que se obtiene durante el proceso de equiparación se denomina *conjunto conflicto*. De todo este conjunto, la etapa 4 selecciona una regla. Esta fase se conoce como *resolución del conjunto conflicto* y estudiaremos diversas técnicas de resolución en la sección 3.5 del tema. La selección de las reglas en la etapa 4 puede verse como un proceso de búsqueda, y ésta puede realizarse sin ningún conocimiento (selección desinformada) o con algún tipo de conocimiento acerca del problema (selección informada). En el primer caso, la selección se hace de un modo totalmente arbitrario. Por ejemplo, seleccionando una regla al azar. En el segundo caso, el mecanismo de control realiza la selección de la regla idónea teniendo en cuenta el conocimiento que se tiene del problema. Finalmente, el algoritmo ejecuta la regla seleccionada (etapa 5) y actualiza la BH con los nuevos hechos resultantes de aplicar la regla (etapa 6).

El coste computacional de un SBR es el resultado de la suma del coste de control (etapas 3 y 4) y del coste de aplicación de las reglas (etapas 5 y 6). Una selección completamente desinformada tiene un pequeño coste de control, ya que la selección arbitraria de la regla no implica gran cantidad de cálculos. Sin embargo, en este caso el coste de aplicación de las reglas es elevado, ya que se requiere ensayar un gran número de reglas para encontrar una solución. Por otra parte, una selección con abundante conocimiento sobre el problema de interés implica un alto coste, tanto en espacio de almacenamiento como en tiempo computacional. Sin embargo, el coste de aplicación de la regla es mínimo, ya que la selección informada guía a una solución de forma

directa. Generalmente, el diseño de un SBR eficiente busca el equilibrio entre ambos costes.

Es importante señalar que, en principio, el mecanismo de inferencias es un algoritmo independiente del conocimiento almacenado en la BC. Por tanto, un SBR desarrollado para una aplicación se puede usar nuevamente en otra aplicación sin más que sustituir la BC. En resumen, se puede decir que el mecanismo de inferencias es el núcleo del SBR, de modo que, considerando los contenidos de la BH, construye dinámicamente una solución, indicando las reglas que deben dispararse y el orden de ejecución.

3.3 Inferencia

Los SBR actuales usan, fundamentalmente, dos modos de inferencia: el encadenamiento hacia delante y el encadenamiento hacia atrás. También es posible encontrar SBR que usan modos de razonamiento mixtos. Comenzaremos estudiando los dos primeros modos de inferencia y al final de la sección, presentaremos el concepto de reversibilidad.

El encadenamiento hacia delante o razonamiento progresivo comienza con una colección de hechos o afirmaciones de partida, y aplica las reglas de la BC repetidas veces hasta que no se generen nuevos hechos. Una regla se puede aplicar si todas las cláusulas del antecedente de la regla se verifican, teniendo en cuenta el contenido de la BH. Cada vez que se aplica una regla se almacenan en la BH los resultados de las acciones del consecuente de dicha regla. El encadenamiento hacia atrás o razonamiento regresivo parte de un conjunto de hipótesis e intenta verificar estas hipótesis usando datos de la BH o datos externos (obtenidos, por ejemplo, a partir del usuario). Si el número de reglas no es muy grande, uno puede obtener un grafo dirigido con las reglas (llamado *red de inferencia*), en el cual las cláusulas del antecedente son las entradas a los nodos y las acciones del consecuente son las salidas de los nodos que, a su vez, pueden ser las cláusulas del antecedente de otros nodos. En esta red, el antecedente que no es consecuente de ninguna otra regla de la red, se convierte en los hechos de partida; y el consecuente, que no es antecedente de ninguna otra regla, se convierte en la meta a alcanzar por el sistema. La Figura 3.2 muestra un ejemplo de red de inferencia con un conjunto de reglas y hechos simbólicos. La forma de representar gráficamente dicha red está tomada de [Winston, 1994]. En ella se representan los hechos o elementos de la BH por medio de las letras que van desde la *a* hasta la *q*; aunque, en general, los antecedentes suelen ser más complejos. En la figura se muestra una red con las siguientes cinco reglas, representadas gráficamente mediante puertas de implicación:

- Regla 1: $a \& b \rightarrow p$
- Regla 2: $b \& c \rightarrow m$
- Regla 3: $d \& e \& f \rightarrow n$
- Regla 4: $n \& g \rightarrow m$
- Regla 5: $h \& m \rightarrow q$

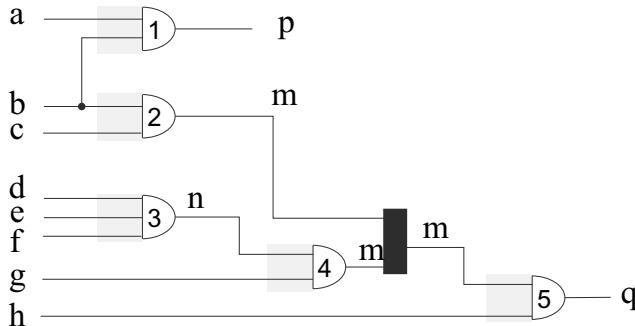


Figura 3.2: Ejemplo de una red de inferencia.

Las reglas 2 y 4 tienen el mismo consecuente, m, que se corresponde con el caso en que se llega a una misma conclusión, siguiendo dos líneas de razonamiento independientes. Esto se representa por medio de un rectángulo negro. A partir de esta red de inferencia, hay dos posibles formas de razonamiento, que consisten en:

- Buscar el conjunto de objetivos que se verifican a partir de un conjunto de entradas. Es lo que se conoce como *encadenamiento hacia delante*. En este tipo de razonamiento, la inferencia progresiva en la red de izquierda a derecha.
- Determinar si se verifica un objetivo dado con ciertas entradas. Es lo que se conoce como *encadenamiento hacia atrás*. Aquí, la inferencia progresiva en la red de derecha a izquierda.

3.3.1 Encadenamiento hacia delante

Aplicando el algoritmo general del motor de inferencias (véase Algoritmo 3.1) al caso particular de un encadenamiento hacia delante, el proceso de inferencias se instanciará en el Algoritmo 3.2. Como puede verse, la particularidad de este algoritmo es la etapa 3 de equiparación, en donde se seleccionan las reglas cuyos antecedentes se verifican, teniendo en cuenta el contenido de la BH.

Algoritmo 3.2 Algoritmo de encadenamiento hacia delante.

```

1: BH = HechosIniciales, ConjuntoConflictos = ExtraeCualquierRegla(BC);
2: mientras NoContenida(Meta,BH) y NoVacío(ConjuntoConflictos) hacer
3:   ConjuntoConflictos = Equiparar(Antecedente(BC),BH);
4:   si NoVacío(ConjuntoConflictos) entonces
5:     R=Resolver(ConjuntoConflictos);
6:     NuevosHechos = Aplicar(R,BH);
7:     Actualizar(BH,NuevosHechos);
8:   fin si
9: fin mientras
10: si Contenida(Meta,BH) entonces
11:   devolver "éxito";
12: fin si
```

Veamos a continuación un ejemplo de aplicación de dicho algoritmo a una red de inferencia descrita por las reglas simbólicas del siguiente ejemplo. Dichas reglas se han tomado de [Borrajo y otros, 1993].

3.3.1.1 Ejemplo de aplicación del encadenamiento hacia delante.

Supongamos que disponemos de la siguiente BC:

- Regla 1: $b \& d \& e \rightarrow f$
- Regla 2: $d \& g \rightarrow a$
- Regla 3: $c \& f \rightarrow a$
- Regla 4: $b \rightarrow x$
- Regla 5: $d \rightarrow e$
- Regla 6: $a \& x \rightarrow h$
- Regla 7: $c \rightarrow d$
- Regla 8: $x \& c \rightarrow a$
- Regla 9: $x \& b \rightarrow d$

La red de inferencia correspondiente a este conjunto de reglas puede verse en la Figura 3.3.

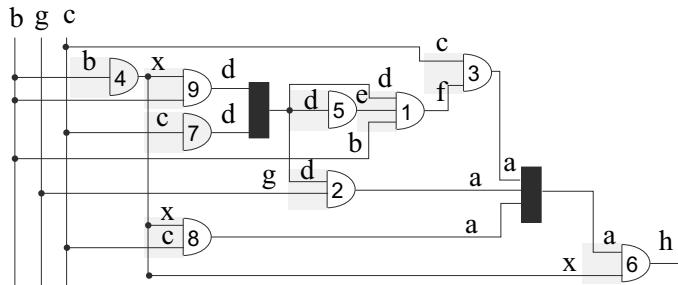


Figura 3.3: Red de inferencia para el ejemplo de aplicación de encadenamiento hacia delante.

Supongamos que partimos de los siguientes contenidos en la BH:

$$\begin{aligned} \text{BH: } & \{b, c\} \\ \text{Meta a alcanzar: } & h \end{aligned}$$

Para aplicar el algoritmo vamos a suponer que una vez que se dispara una regla ya no se puede volver a disparar. Esta suposición es correcta en este ejemplo particular (pero no en general), ya que la ejecución repetida de una regla no modifica en nada la BH (en cada repetición se genera un hecho que ya está contenido en la BH desde la primera vez que se ha disparado la regla). Con esta suposición evitamos que una

regla esté continuamente ejecutándose sin producir modificaciones en la BH (esta aproximación práctica se conoce como *principio de refracción*).

Con respecto a la forma de selección de las reglas (etapa 5 del Algoritmo 3.2), vamos a suponer dos estrategias distintas:

- **Estrategia 1:** Seleccionar la regla de menor número. Inicialmente las únicas reglas que se pueden aplicar en el ciclo 1 son

$$\text{ConjuntoConflicto} = \{4, 7\}$$

tal y como se muestra en la Figura 3.4. A continuación, se selecciona y aplica la Regla 4 (la de menor número). La BH se actualiza a

$$\text{BH: } \{x, b, c\}$$

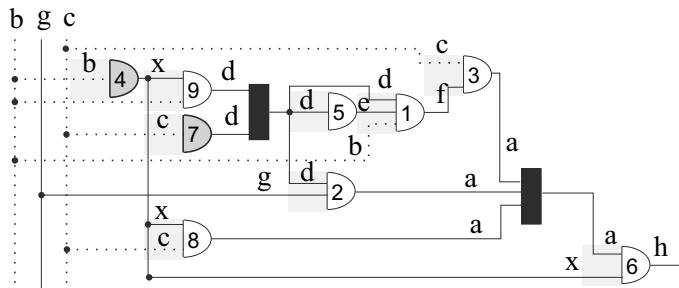


Figura 3.4: Red de inferencia en el ciclo 1: el conjunto conflicto está formado por las reglas 4 y 7 (resaltadas en sombreado gris) y la BH contiene los hechos b y c (líneas discontinuas).

En el ciclo 2, las únicas reglas a aplicar son:

$$\text{ConjuntoConflicto} = \{7, 8, 9\}$$

Esto se muestra en la Figura 3.5. Se selecciona y aplica la Regla 7, y se actualiza la BH a

$$\text{BH: } \{d, x, b, c\}$$

En el tercer ciclo, las reglas que se pueden aplicar son las siguientes, tal y como se puede apreciar en la Figura 3.6:

$$\text{ConjuntoConflicto} = \{5, 8, 9\}$$

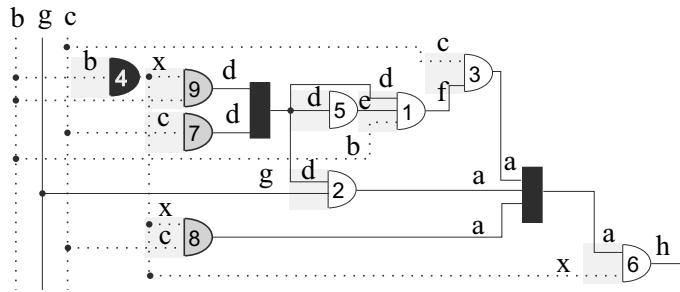


Figura 3.5: Red de inferencia en el ciclo 2: la regla 4 ya ha sido disparada (resaltada en negro) y el conjunto conflicto está formado por las reglas 7, 8 y 9 (resaltadas en gris).

Se selecciona la Regla 5, se aplica y la BH se actualiza a

$$\text{BH: } \{e, d, x, b, c\}$$

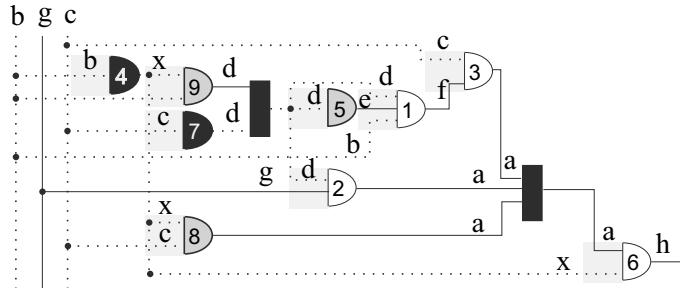


Figura 3.6: Red de inferencia en el ciclo 3: las reglas 4 y 7 han sido disparadas previamente, y el conjunto conflicto está formado por las reglas 5, 8 y 9.

En el ciclo 4, tal y como se muestra en la Figura 3.7:

$$\text{ConjuntoConflict} = \{1, 8, 9\}$$

Se selecciona la Regla 1, se aplica y la BH se actualiza a

$$\text{BH: } \{f, e, d, x, b, c\}$$

En el ciclo 5, tal y como se muestra en la Figura 3.8:

$$\text{ConjuntoConflict} = \{3, 8, 9\}$$

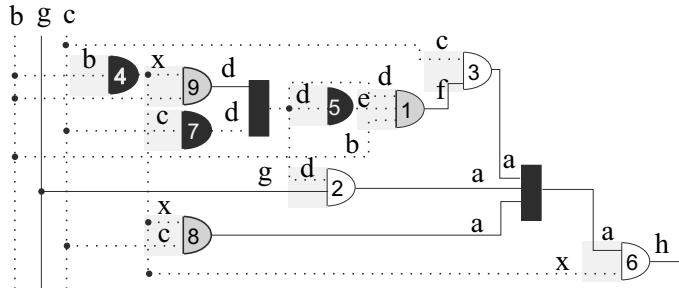


Figura 3.7: Red de inferencia en el ciclo 4: las reglas 4, 5 y 7 han sido disparadas previamente, y el conjunto conflicto está formado por las reglas 1, 8 y 9.

Se selecciona la Regla 3, se aplica y la BH se actualiza a

$$\text{BH: } \{a, f, e, d, x, b, c\}$$

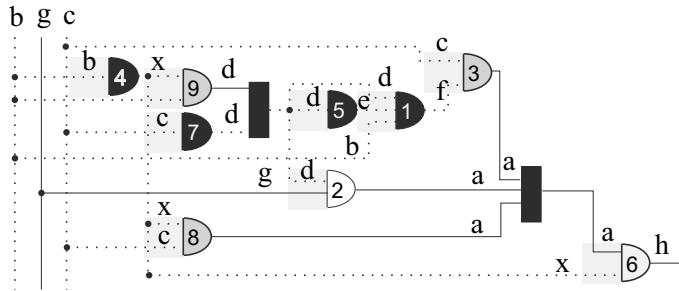


Figura 3.8: Red de inferencia en el ciclo 5: las reglas 1, 4, 5 y 7 han sido disparadas previamente, y el conjunto conflicto está formado por las reglas 3, 8 y 9.

Por último, tal y como se muestra en la Figura 3.9:

$$\text{ConjuntoConflicto} = \{6, 8, 9\}$$

Se selecciona la Regla 6, se aplica y la BH se actualiza a

$$\text{BH: } \{h, a, f, e, d, x, b, c\}$$

Como la meta h está contenida en la BH, finaliza el proceso de inferencias alcanzando el éxito. En la Tabla 3.1 se muestra todo este proceso de aplicación de la estrategia 1.

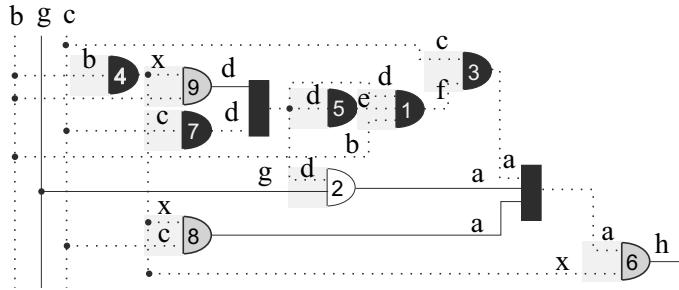


Figura 3.9: Red de inferencia en el ciclo 6: las reglas 1, 3, 4, 5, y 7 han sido disparadas y el conjunto conflicto está formado por las reglas 6, 8 y 9.

Ciclo	BH	Conjunto Conflicto	Resolución
1	$\{b, c\}$	$\{4, 7\}$	4
2	$\{x, b, c\}$	$\{7, 8, 9\}$	7
3	$\{d, x, b, c\}$	$\{5, 8, 9\}$	5
4	$\{e, d, x, b, c\}$	$\{1, 8, 9\}$	1
5	$\{f, e, d, x, b, c\}$	$\{3, 8, 9\}$	3
6	$\{a, f, e, d, x, b, c\}$	$\{6, 8, 9\}$	6

Tabla 3.1: Aplicación de la estrategia 1 a la red de inferencia de la Figura 3.3.

- **Estrategia 2:** Seleccionar la regla que tiene mayor número de cláusulas antecedentes (es decir, la más restrictiva). Entre dos o más reglas con igual número de antecedentes, se elegirá la regla de menor número. Esta segunda estrategia se plantea para clarificar cómo la elección de una estrategia puede influir en el número de ciclos que dura el proceso de encadenamiento.

Partiendo de la red de inferencia de la Figura 3.3 y la BH inicial

$$\text{BH: } \{b, c\}$$

$$\text{Meta a alcanzar: } h$$

en el primer ciclo, tal y como muestra la Figura 3.4:

$$\text{ConjuntoConflicto} = \{4, 7\}$$

Ya que tanto la Regla 4 como la 7 tienen una única cláusula antecedente, se selecciona y aplica la Regla 4 (la de menor número). La BH se actualiza a

$$\text{BH: } \{x, b, c\}$$

En el ciclo 2, las posibles reglas a aplicar son (véase la Figura 3.5):

$$\text{ConjuntoConflict}=\{7, 8, 9\}$$

Las reglas más restrictivas, ambas con dos cláusulas, son la 8 y la 9. Se selecciona y aplica la 8 (menor número), y se actualiza la BH a

$$\text{BH: } \{a, x, b, c\}$$

En el tercer ciclo, las reglas que se pueden aplicar son las siguientes, tal y como se puede apreciar en la Figura 3.10:

$$\text{ConjuntoConflict}=\{6, 7, 9\}$$

Las reglas más restrictivas son la 6 y la 9. Se selecciona la Regla 6, se aplica y la BH se actualiza, de tal forma, que pasa a contener la meta. Por tanto, en este caso también se acabaría el proceso de inferencias con éxito. En la Tabla 3.2 se muestra todo este proceso de aplicación de la estrategia 2.

$$\text{BH: } \{h, a, x, b, c\}$$

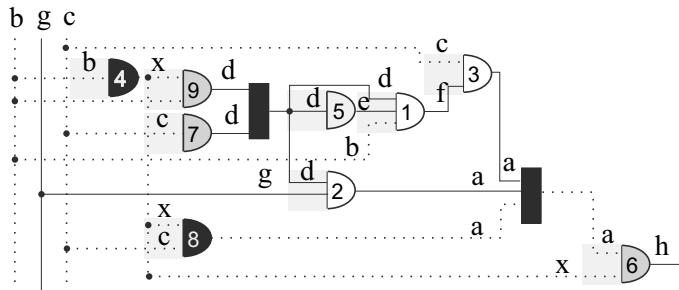


Figura 3.10: Red de inferencia en el ciclo 3 (al aplicar la segunda estrategia): las reglas 4 y 8 han sido disparadas previamente, y el conjunto conflicto está formado por las reglas 6, 7 y 9.

Comparando la aplicación de las dos estrategias, hay que señalar que se necesitan 6 ciclos para alcanzar la meta con la primera estrategia, mientras que solamente 3 con la segunda. Como vemos, en este segundo caso el coste de aplicación de las reglas se reduce a la mitad. Sin embargo, el coste de selección de las reglas es mayor.

Siguiendo la primera estrategia, no hace falta obtener todo el conjunto conflicto; bastará con buscar la primera regla que verifique todas las cláusulas de sus antecedentes. Una vez encontrada, se parará el proceso de búsqueda y se aplicará la regla.

Ciclo	BH	Conjunto Conflicto	Resolución
1	$\{b, c\}$	$\{4, 7\}$	4
2	$\{x, b, c\}$	$\{7, 8, 9\}$	8
3	$\{a, x, b, c\}$	$\{6, 7, 9\}$	6

Tabla 3.2: Aplicación de la estrategia 2 a la red de inferencia de la Figura 3.3.

No obstante, dicha estrategia sólo se debe de aplicar si el mecanismo de inferencias puede retroceder sobre sus pasos cuando llega a una situación en la cual no puede aplicar más reglas, y aún quedan por explorar otras alternativas.

Por otra parte, el proceso de selección de la regla más restrictiva obtiene el conjunto de todas las reglas que se pueden aplicar en cada ciclo, y sobre ese conjunto selecciona la más restrictiva. Este proceso requiere más tiempo en computación que el anterior.

3.3.1.2 Ciclo de reconocimiento-acción

El proceso iterativo de aplicación del encadenamiento hacia delante se denomina ciclo de reconocimiento-acción (véase Figura 3.11). Durante la fase de equiparación el mecanismo de inferencias examina la BH y selecciona el conjunto de reglas cuya parte antecedente es compatible con la BH. Esta operación es relativamente sencilla cuando las reglas no contienen variables, pero se vuelve más compleja cuando se permite la utilización de variables y cuantificadores en las reglas. Este problema se estudiará en la siguiente sección cuando hablaremos del método RETE, un método de equiparación que permite realizar este proceso de forma eficiente.

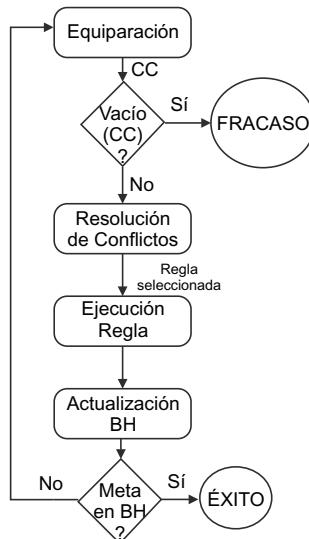


Figura 3.11: Ciclo de reconocimiento-acción (CC es el conjunto conflicto).

Todas las reglas para las cuales la equiparación ha tenido éxito son válidas y forman el conjunto conflicto. La fase de resolución de conflictos selecciona a partir del conjunto conflicto la regla que se disparará. Generalmente, los SBR sólo permiten que en cada ciclo se dispare una única regla. Una vez elegida la regla, el mecanismo de inferencias aplica la regla sobre la BH, es decir, activa la parte de acción de la regla, la cual añade o elimina hechos de la BH. Adicionalmente, los SBR reales suelen permitir la ejecución de procedimientos sobre la parte de acción y, a veces, sobre la de condición. Dichos procedimientos proporcionan una vía para disparar tareas anexas como entradas-salidas, cálculos matemáticos, etc.

El ciclo de reconocimiento-acción se repite hasta que el hecho que se ha fijado como meta se añade a la BH (en cuyo caso se alcanza la meta buscada con éxito) o cuando ya no puede aplicarse ninguna regla más (en cuyo caso se fracasa en la obtención de la meta).

Los inconvenientes de este método de encadenamiento proceden de la falta de focalización hacia la meta buscada. Podemos citar los siguientes:

1. La fase de resolución de conflictos es crítica.
2. El disparo de las reglas elegidas, a pesar de que las conclusiones a las que llegan no tengan interés, favorece la explosión combinatoria.
3. Se necesita cargar la BH con todas las afirmaciones que se poseen, sin saber si los elementos que se introducen serán útiles o no, o si faltan datos.

3.3.2 Encadenamiento hacia atrás

En un modo de razonamiento con encadenamiento hacia atrás, se especifica una meta objetivo y se trata de determinar si ese objetivo se verifica o no, teniendo en cuenta el contenido de la BH. Esto puede verse en el Algoritmo 3.3, el cual hace una llamada al procedimiento *Verificar*, descrito en el Algoritmo 3.4. En dicho procedimiento, se investigan los consecuentes de todas las reglas, con el fin de encontrar aquellas cuyos consecuentes contengan el objetivo a verificar. El subconjunto de reglas resultantes se examina para descubrir alguna que verifique todos sus antecedentes, teniendo en cuenta los contenidos de la BH. Si existe tal regla, entonces se verifica el objetivo; en caso contrario, los antecedentes que no se pudieron verificar pasan a ser nuevos objetivos a verificar, en un control recursivo. Este tipo de búsqueda se corresponde con una búsqueda en profundidad (véase el capítulo 8).

Algoritmo 3.3 Algoritmo de encadenamiento hacia atrás.

- 1: BH = HechosIniciales;
 - 2: **si** Verificar (Meta,BH) **entonces**
 - 3: devolver "éxito";
 - 4: **si no**
 - 5: devolver "fracaso";
 - 6: **fin si**
-

Algoritmo 3.4 Procedimiento Verificar: comprueba si existe un conjunto de reglas verificando una meta.

```

1: Verificado=Falso;
2: si Contenida (Meta,BH) entonces
3:   devolver Verdadero;
4: si no
5:   ConjuntoConflict = Equiparar(Consecuentes(BC),Meta);
6:   mientras NoVacío(ConjuntoConflict) y No(Verificado) hacer
7:     R=Resolver(ConjuntoConflict);
8:     Eliminar(R,ConjuntoConflict);
9:     NuevasMetas=ExtraerAntecedentes(R), Verificado=Verdadero;
10:    mientras NoVacío(NuevasMetas) y Verificado hacer
11:      Meta=SeleccionarMeta(NuevasMetas);
12:      Eliminar(Meta, NuevasMetas);
13:      Verificado=Verificar(Meta,BH);
14:      si Verificado entonces
15:        Añadir(Meta,BH);
16:      fin si
17:    fin mientras
18:  fin mientras
19:  devolver(Verificado);
20: fin si
```

Veamos ahora, en detalle, el procedimiento *Verificar* (véase Algoritmo 3.4). Primero, se comprueba si la meta propuesta está contenida en la BH (etapa 2). Si no es así, intenta demostrarlo utilizando las reglas que posee (etas 5 en adelante). La etapa 5 obtiene el conjunto conflicto de todas las reglas que tienen la meta en su parte de acción. El algoritmo intenta encontrar, entre éstas, alguna que verifique la meta buscada (bucle de la etapa 6). Este es un proceso disyuntivo: si se encuentra alguna regla que permite demostrar la meta, devuelve el valor "verdadero". Por otra parte, una regla demuestra una meta si todos sus antecedentes se verifican (bucle de la etapa 10). Este es un proceso conjuntivo: si todos los antecedentes se verifican, se devuelve el valor "verdadero". Por último, el procedimiento Verificar es recursivo, tal y como se puede apreciar en la etapa 13.

Este algoritmo propuesto está muy simplificado. Si un hecho no está en la BH o no se puede deducir a partir de la BC, entonces se considera que no se verifica. Los sistemas reales no realizan esta suposición. En vez de ello, disponen de mecanismos necesarios para intentar obtener esa información desde fuentes externas al sistema (preguntando al usuario, a través de sensores o accediendo a bases de datos externas).

3.3.2.1 Ejemplo de aplicación del encadenamiento hacia atrás

Retomemos la BC representada por la red de inferencia de la Figura 3.3, la misma BH inicial que en el ejemplo del encadenamiento hacia delante y el mismo criterio de resolución del conjunto conflicto (es decir, la regla de menor número).

BH: $\{b, c\}$; Meta a alcanzar: h

A diferencia del encadenamiento hacia delante, ahora se recorre la red de inferencia de derecha a izquierda. Veámoslo. En el primer ciclo, la única regla que contiene en su parte consecuente la meta h es la Regla 6. Para verificar que se puede aplicar dicha regla, en el segundo ciclo, las nuevas metas a considerar serán las cláusulas del antecedente de dicha regla, es decir,

$$\text{Nuevas Metas: } \{a, x\}$$

Comenzaremos revisando la meta a . Esta puede alcanzarse vía tres reglas distintas (véase Figura 3.12):

$$\text{ConjuntoConflictos} = \{2, 3, 8\}$$

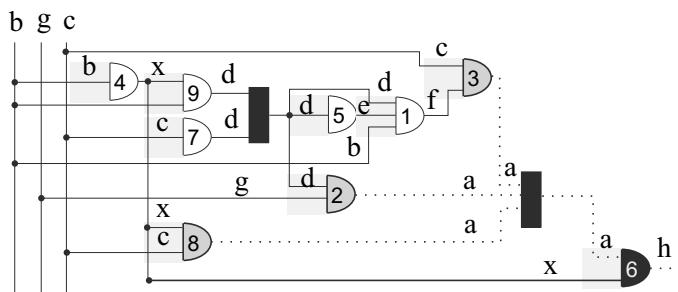


Figura 3.12: Red de inferencia en el ciclo 2, en la que se intenta determinar si el antecedente a de la regla 6 se verifica. Así, el conjunto conflicto está formado por las reglas 2, 3 y 8.

Aplicando el criterio de selección por orden de numeración, se selecciona primero la Regla 2. En el tercer ciclo, para alcanzar la meta a aplicando la Regla 2, se necesita que se verifiquen conjuntamente las dos cláusulas de su antecedente:

$$\text{Nuevas Metas: } \{d, g\}$$

Con el fin de verificar la meta d , se obtiene (véase Figura 3.13):

$$\text{ConjuntoConflictos} = \{7, 9\}$$

Si intentamos verificar d aplicando la Regla 7, debemos comprobar si su antecesor c está en la BH. Como es así, d pasa a estar verificado y se añade a la BH:

$$\text{BH: } \{d, b, c\}$$

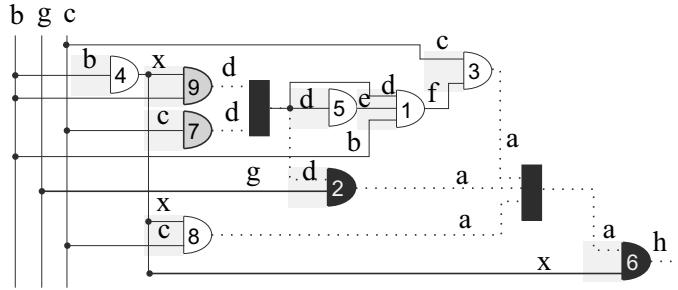


Figura 3.13: Red de inferencia en el ciclo 3, en la que se intenta determinar si el antecedente d de la regla 2 se verifica. Así, el conjunto conflicto está formado por las reglas 7 y 9.

En el ciclo 4, se comprueba si se verifica g (véase Figura 3.14). Como g no está en la BH, la Regla 2 no se puede verificar. En el ciclo 5, será necesario probar con otras reglas para comprobar a . Ahora (véase Figura 3.15)

$$\text{ConjuntoConflict} = \{3, 8\}$$

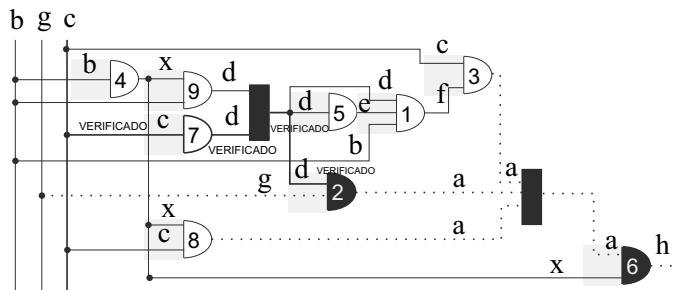


Figura 3.14: Red de inferencia en el ciclo 4, en la que se intenta determinar si el antecedente g de la regla 2 se verifica.

Se selecciona la Regla 3 y exploran las cláusulas de su antecedente. El primero es c , que está contenido en la BH. El segundo es f , cuya verificación implica comprobar las cláusulas del antecedente de la Regla 1 en el ciclo 6, puesto que es la única que lo tiene como consecuente. Los antecedentes de f son d , que ya está verificado, e , que implica verificar la regla 5 en el ciclo 7, y b , que también está en la BH. En el ciclo 7, se comprueba que el antecedente d de la regla 5 se verifica, por lo que su consecuente e también lo hará. Se actualiza, así, la BH a:

$$\text{BH: } \{e, d, b, c\}$$

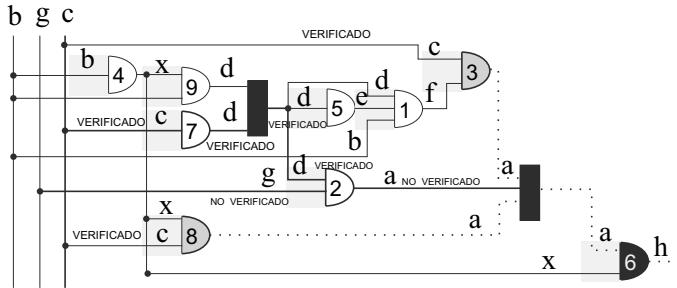


Figura 3.15: Red de inferencia en el ciclo 5, en la que se intenta determinar si el antecedente a se verifica.

Como consecuencia de ello, la Regla 1 se verifica y su consecuente f se añade a la BH:

$$\text{BH: } \{f, e, d, b, c\}$$

Por tanto, se verifican todos los antecedentes de la Regla 3 y se añade su consecuente a a la BH:

$$\text{BH: } \{a, f, e, d, b, c\}$$

Por último, para verificar la meta h , se necesita comprobar la validez del hecho x . En el ciclo 8,

$$\text{ConjuntoConflictos} = \{4\}$$

Se selecciona la Regla 4 y la nueva meta es su antecedente b en el ciclo 9. Como dicho antecedente está contenido en la BH, se verifica el hecho x y finaliza de forma exitosa la aplicación del encadenamiento hacia atrás:

$$\text{BH} = \{h, x, a, f, d, b, c\}$$

3.3.2.2 Ciclo de encadenamiento hacia atrás

En el encadenamiento hacia atrás el ciclo de inferencias se convierte en:

1. *Etapa de equiparación:* con búsqueda de las reglas cuya conclusión se corresponde con la meta M en curso.
2. *Etapa de resolución de conflictos:* con selección de una regla entre las dadas por el conjunto conflicto.

3. *Etapa de acción:* con reemplazamiento de la meta M por la conjunción de las condiciones del antecedente de la regla seleccionada.

El ciclo finaliza cuando la meta inicial se ha reducido a submetas elementales verificadas en la BH o cuando no se ha podido encontrar ninguna regla que llegue a alguna submeta válida. El problema de este modo de razonamiento es entrar en bucles infinitos, en los cuales una meta es, a su vez, una submeta en su propio árbol de búsqueda. Por ejemplo, si para demostrar A es necesario demostrar B y para demostrar B se necesita demostrar A. Aunque no vamos a entrar en ello, debemos comentar que existen procedimientos para detectar dichos bucles y eliminar sus efectos.

El encadenamiento hacia atrás tiene una serie de ventajas con respecto al encadenamiento hacia delante:

1. El sistema sólo planteará cuestiones al usuario cuando tiene necesidad de ello, es decir, después de haber explorado todas las posibilidades de la BC.
2. Limita el número de equiparaciones de antecedentes de las reglas, ya que sólo se equiparan aquellas cuyo consecuente se requiere verificar.
3. Disminuye la dimensión del árbol de búsqueda, como consecuencia de la limitación en las equiparaciones.

3.3.3 Reversibilidad

Ahora prodríamos preguntarnos si, dado un problema concreto, es lo mismo usar un modo de razonamiento con encadenamiento hacia delante o hacia atrás. La respuesta es que no, ya que puede ser significativamente más eficiente buscar en una dirección que en otra. El uso de un tipo u otro depende del problema concreto. Aquí sólo podemos indicar un conjunto de estrategias generales, basadas en la forma de la red de inferencia y en el grado de conocimiento que el usuario del sistema tiene de los datos y de los objetivos del problema.

1. Si la red tiene, en promedio, un número elevado de reglas con muchas condiciones en el antecedente, se debe elegir un encadenamiento hacia delante; mientras que si hay muchas reglas cuyo consecuente forma parte del antecedente de otras muchas, entonces se debe elegir un encadenamiento hacia atrás.
2. En muchos problemas conocemos perfectamente la naturaleza de los datos de entrada, mientras que no está tan claro cuáles son los objetivos a alcanzar. En este caso, es preferible un encadenamiento hacia delante, también conocido por ello como *razonamiento conducido por los datos*. Por otra parte, si los objetivos y subobjetivos están bien definidos, pero no las estructuras de los datos, es preferible un encadenamiento hacia atrás, también conocido como *razonamiento conducido por objetivos*. Por ejemplo, supongamos un SBR diseñado para ayudar a realizar la compra on-line en un tienda especializada. Las entradas al sistema son las preferencias y las características del cliente y la salida es el artículo recomendado. Un encadenamiento hacia delante accederá mejor a los datos si

el usuario del sistema es el propio cliente, ya que éste conoce sus preferencias y características y lo que necesita es consejo sobre lo que puede comprar en la tienda. Si la función del sistema es aconsejar al vendedor para que éste sugiera al cliente, es más aconsejable un encadenamiento hacia atrás, ya que el vendedor conoce mejor las características de los artículos de la tienda, pero no está muy seguro de las preferencias y características del cliente.

3. Un tipo de razonamiento conducido por los datos es apropiado para problemas que requieren tareas reactivas como, por ejemplo, la monitorización; y un tipo de razonamiento conducido por objetivos es aconsejable para problemas que se basan en tareas analíticas como, por ejemplo, la clasificación.
4. Los SBR que deban justificar su razonamiento, deben proceder en la dirección que se adapta más a la manera de pensar del usuario del sistema. Por ejemplo, un SBR para diagnóstico en medicina suele ser más aceptado por los médicos si es capaz de explicar el consejo que proporciona, y, para ello, debe razonar hacia atrás.

Además de los mecanismos de razonamiento hacia delante y hacia atrás, existe otra posibilidad que consiste en razonar siguiendo ambos modos de razonamiento a la vez por la misma de red de inferencia, hasta que los dos caminos de búsqueda se encuentran. Este modo de razonamiento mixto se llama *búsqueda bidireccional* o *reversibilidad*. Para aplicar este tipo de razonamiento, se requiere:

1. Incorporar a la BH global tanto las descripciones de los datos iniciales como de las metas a alcanzar.
2. Definir cuidadosamente las reglas, distinguiendo entre las reglas aplicables en el encadenamiento hacia delante (reglas tipo A) y en el encadenamiento hacia atrás (reglas tipo B).
3. Diseñar la condición de terminación de la inferencia para que considere tanto la parte de descripción de los datos inferidos como de las metas a alcanzar.
4. Modelar el mecanismo de control para que decida en cada etapa si aplica una regla de tipo A o una regla de tipo B.

Esta búsqueda parece ser adecuada si el número de nodos en cada paso crece exponencialmente con el número de pasos que se han dado. Resultados empíricos sugieren que, para búsquedas completamente desinformadas, esta estrategia basada en *divide y vencerás* es efectiva; mientras que, para búsquedas informadas, no suele ser así.

Por último, resaltar que uno de los problemas que tiene esta búsqueda es que el encadenamiento hacia delante pueda explorar una parte de la red de inferencia distinta de la parte investigada por el encadenamiento hacia atrás. Para evitar esto, es necesario diseñar el sistema cuidadosamente para que llegue a usar cada forma de razonamiento en las situaciones más aconsejables.

3.4 Técnicas de equiparación

La equiparación del antecedente de las reglas con el estado de la BH no siempre es obvia, tal y como hemos visto en los ejemplos anteriores, ya que el antecedente puede no describir situaciones particulares sino generales. Así, un tipo sencillo de equiparación surge cuando dicho antecedente contiene variables. Esto lo veremos en la subsección 3.4.1.

Otro problema, relacionado con la equiparación en el algoritmo general del motor de inferencias (véase Algoritmo 3.1), es la necesidad de examinar todas las reglas en cada ciclo de inferencias. Este proceso es poco eficiente, si hay que recorrer toda la BC y ésta contiene numerosas reglas. Se puede simplificar mediante:

1. **Técnicas de indexación:** Consisten en añadir a las reglas nuevas condiciones relacionadas con el punto de inferencia. Este recurso permite dividir el problema en varias etapas y agrupar las reglas en función de la etapa en la que se aplican. Por ejemplo, las reglas de un SBR en medicina prodrían contener una cláusula objetivo que indicase la tarea a realizar: recopilación de antecedentes, diagnóstico, recomendación de tratamiento, aplicación del tratamiento, etc. Lamentablemente, esta forma de indexar las reglas sólo se puede aplicar si las condiciones de las reglas se equiparan exactamente con la BH. Además, con esta aproximación se pierde generalidad en la declaración de las reglas. A pesar de todo, la indexación suele ser un factor importante para la eficiencia de los SBR.
2. **Técnicas que aceleran el proceso de equiparación,** sin necesidad de examinar toda la BC. El método más conocido es el algoritmo RETE, que estudiaremos en la subsección 3.4.2.

3.4.1 Equiparación con variables

Generalmente, el antecedente de las reglas describe situaciones generales que requieren ser expresadas mediante variables. Nos centraremos primero en ver cómo representar reglas con variables. A continuación, estudiaremos las diferentes situaciones que surgen cuando se equiparan las reglas expresadas mediante variables con la BH.

3.4.1.1 Sintaxis de reglas con variables

Aquí propondremos una *sintaxis* basada en CLIPS¹[Calvo y otros, 2005; Steele, 1990], aunque ésta siempre puede variar en función del lenguaje o la herramienta que utilicemos. Para clarificar la sintaxis que proponemos, vamos a considerar nuevamente la siguiente regla, expresadas en lenguaje natural:

Paciente menor de 10 años con manchas rojas y fiebre → Paciente con varicela

¹CLIPS (C Language Integrated Production System) es una herramienta de libre distribución para desarrollar sistemas expertos. <http://www.ghg.net/clips/CLIPS.html>

La sintaxis propuesta es la siguiente:

1. Cada elemento de condición presente en el antecedente de una regla debe ir encerrado entre paréntesis y empezar por una constante.
2. Todos los elementos de condición del antecedente de una regla deben ir unidos por el operador lógico Y (representado por el símbolo \wedge).
3. Los elementos de condición están formados por átomos, donde un átomo puede ser una constante o una variable. Para distinguir las variables de las constantes, es usual señalar las primeras con un prefijo de interrogación. Así, la representación de las condiciones de la regla ejemplo incluiría:
 - (paciente $?p$) para expresar el paciente $?p$.
 - (edad $?p ?e$) para indicar que la edad del paciente $?p$ es $?e$.
 - (síntoma $?p$ fiebre) para representar que el paciente $?p$ presenta el síntoma fiebre.
4. Los elementos de condición pueden expresar algún tipo de prueba sobre las variables. En el ejemplo, para indicar que el paciente debe ser menor de 10 años, vamos a utilizar la palabra clave *test*:

(test ($<?e 10$))

5. Las acciones del consecuente también pueden contener variables. Para la regla ejemplo, la acción del consecuente que expresa que el paciente padece varicela, se podría representar como

(enfermedad $?p$ varicela)

6. Las acciones del consecuente pueden incluir dos predicados: Añadir (para agregar nuevos hechos a la BH) y Borrar (para eliminar hechos existentes de la BH). En el ejemplo,

Añadir(enfermedad $?p$ varicela)

La representación completa de la regla ejemplo, siguiendo esta sintaxis será la siguiente:

SI (paciente $?p$) \wedge (edad $?p ?e$) \wedge (síntoma $?p$ fiebre) \wedge (síntoma $?p$ manchas-rojas)
ENTONCES Añadir(enfermedad $?p$ varicela)

3.4.1.2 Equiparación de reglas utilizando variables con los contenidos de la BH

La equiparación de reglas usando variables es un caso particular del problema de la unificación de términos en la resolución de primer orden, del que se habla en el capítulo 2. Teniendo en cuenta la sintaxis presentada, veamos ahora cómo se equiparan las reglas que incluyen variables con la BH. Como ejemplo, imaginemos un SBR para un hipódromo con los siguientes contenidos en la BH:

1. Hay cuatro caballos: Cometa, Bronco, Veloz y Rayo.
2. La relación entre ellos es la siguiente: Cometa es padre de Veloz y Bronco, y Veloz es padre de Rayo.
3. Dos de ellos son muy rápidos: Bronco y Rayo.

Los datos de este ejemplo están tomados de [Winston, 1994]. Primeramente, nos deberemos plantear cómo representar la BH. Una opción sería la siguiente:

- (caballo ?x) para expresar el caballo ?x.
- (padre ?x ?y) para indicar que el caballo ?x es padre de ?y.
- (rápido ?x) para representar que el caballo ?x es rápido.

Los contenidos concretos de la BH inicial serán los siguientes:

(caballo Cometa), (caballo Bronco), (caballo Veloz), (caballo Rayo)
(padre Cometa Veloz), (padre Cometa Bronco), (padre Veloz Rayo)
(rápido Bronco), (rápido Rayo)

Durante la equiparación, pueden darse las siguientes situaciones:

1. *Una variable aparece una sola vez en la parte de condición de una regla.* En este caso, la variable se equipara con cualquier valor que ocupe la misma posición en un elemento de la BH. Por ejemplo, si la regla es

SI (caballo ?x) **ENTONCES**

se equipara con todos los elementos de la BH cuyo valor para la primera posición es *caballo*. En nuestro ejemplo, se equiparía con (caballo Cometa), (caballo Bronco), (caballo Veloz) y (caballo Rayo) para los valores de Cometa, Bronco, Rayo y Veloz para la variable *x*.

2. *Una variable aparece más de una vez en la parte de condición de una regla.* En este caso, la variable debe equiparse siempre con el mismo valor en todas las ocurrencias de la regla. Por ejemplo, si la regla es

SI (caballo ?x) \wedge (rápido ?x) **ENTONCES** (candidato-competición ?x)

la primera condición se equipara con todos los elementos de la BH cuyo valor para la primera posición es *caballo*, y la segunda condición se equipara con todos los elementos de la BH cuyo valor para la primera posición es *rápido*. La regla se equipara con los elementos de la BH que verifiquen el mismo valor para la variable *x* en ambas condiciones. Es decir,

- (a) Equiparación de la primera condición: (caballo Cometa), (caballo Bronco), (caballo Veloz) y (caballo Rayo) para los valores de Cometa, Bronco, Rayo y Veloz para la variable *x*.
 - (b) Equiparación de la segunda condición: (rápido Bronco) y (rápido Rayo) para los valores de Bronco y Rayo para la variable *x*.
 - (c) Equiparación de la regla: (caballo Bronco) \wedge (rápido Bronco) y (caballo Rayo) \wedge (rápido Rayo) para los valores de Bronco y Rayo para la variable *x*.
3. *Varias variables aparecen en la parte de condición de una regla.* Se pueden equiparar variables distintas con un mismo valor. Por ejemplo, si la regla es

SI (caballo ?x) \wedge (padre ?x ?y) \wedge (rápido ?y) **ENTONCES** ...

resultan las siguientes equiparaciones:

- (a) Equiparación de la primera condición: (caballo Cometa), (caballo Bronco), (caballo Veloz) y (caballo Rayo) para los valores de Cometa, Bronco, Rayo y Veloz para la variable *x*.
 - (b) Equiparación de la segunda condición: (padre Cometa Veloz), (padre Cometa Bronco) y (padre Veloz Rayo) para los pares de valores (Cometa, Veloz), (Cometa, Bronco) y (Veloz, Rayo) para las variables *x* e *y*, respectivamente.
 - (c) Equiparación de la tercera condición: (rápido Bronco) y (rápido Rayo) para los valores de Bronco y Rayo para la variable *y*.
 - (d) Equiparación de la primera y segunda condición: (caballo Cometa) \wedge (padre Cometa Veloz), (caballo Cometa) \wedge (padre Cometa Bronco), y (caballo Veloz) \wedge (padre Veloz Rayo) para los pares de valores (Cometa, Veloz), (Cometa, Bronco) y (Veloz, Rayo) para las variables *x* e *y*, respectivamente.
 - (e) Equiparación de la regla: (caballo Cometa) \wedge (padre Cometa Bronco) \wedge (rápido Bronco) y (caballo Veloz) \wedge (padre Veloz Rayo) \wedge (rápido Rayo) para los pares de valores (Cometa, Bronco) y (Veloz, Rayo) para las variables *x* e *y*, respectivamente.
4. *Una o varias variables aparecen en una o varias condiciones negadas de una regla.* La equiparación se da si se cumplen las dos siguientes restricciones:

- (a) Se equiparan todas las condiciones no negadas para algún o algunos elementos de la BH.
- (b) No existe ningún elemento en la BH que pueda equiparar las condiciones negadas.

La aplicación de estas dos restricciones se conoce como *Hipótesis del Mundo Cerrado*: Todo hecho que no esté en la BH, durante la equiparación de una regla, se considera falso. Por ejemplo, si la regla es

SI (caballo ?x) \wedge \neg (rápido ?x) **ENTONCES** (candidato-reserva ?x)

resultan las siguientes equiparaciones:

- (a) Equiparación de la primera condición: (caballo Cometa), (caballo Bronco), (caballo Veloz) y (caballo Rayo) para los valores de Cometa, Bronco, Rayo y Veloz para la variable x .
- (b) Equiparación de la segunda condición sin negar: (rápido Bronco) y (rápido Rayo) para los valores de Bronco y Rayo para la variable x .
- (c) Equiparación de la regla: (caballo Cometa) y (caballo Veloz) para los valores de Cometa y Veloz para la variable x .

Se llama *instanciación* al par formado por una regla y los elementos de la BH que equiparan dicha regla. En el ejemplo anterior, la equiparación ha dado lugar a dos instanciaciones (conjunto conflicto):

1. {**SI** (caballo ?x) \wedge \neg (rápido ?x) **ENTONCES** (candidato-reserva ?x), (?x Cometa)}
2. {**SI** (caballo ?x) \wedge \neg (rápido ?x) **ENTONCES** (candidato-reserva ?x), (?x Veloz)}

En la primera instancia la variable x queda ligada al valor *Cometa*, y en la segunda al valor *Veloz*. Esta ligadura se realiza en la BH y no en la regla. Así, durante la fase de acción, no se ejecuta la regla sino una instancia de ella. Si en el consecuente de una regla aparece la variable (como en el ejemplo), durante la fase de acción la variable se sustituye por su valor ligado en la instancia. En el ejemplo, si se selecciona aleatoriamente una instancia del conjunto conflicto, la ejecución de la primera instancia añadiría un nuevo hecho a la BH, actualizándose ésta al siguiente contenido:

(caballo Cometa), (caballo Bronco), (caballo Veloz), (caballo Rayo)
 (padre Cometa Veloz), (padre Cometa Bronco), (padre Veloz Rayo)
 (rápido Bronco), (rápido Rayo), (candidato-reserva Cometa)

3.4.2 El algoritmo RETE

El algoritmo de equiparación RETE evita examinar todas las reglas de la BC con todos los datos de la BH. Para hacer esto, explota dos propiedades comunes a todos los SBR:

1. Las condiciones de las reglas contienen muchos patrones similares, aunque no idénticos. RETE agrupa estos patrones comunes con el fin de evitar la equiparación de patrones idénticos repetidas veces. La agrupación de patrones comunes la realiza un compilador antes de la ejecución del SBR. Dicho compilador genera un grafo llamado red RETE, que es una red de clasificación estructurada en forma de árbol. Los nodos de la red indican las operaciones que se realizan en la equiparación, y los arcos indican el camino de nodo a nodo por el que deben de fluir los datos. Se eliminan las operaciones redundantes construyendo un único nodo por cada operación distinta a realizar y uniendo dicho nodo con todos los demás nodos que requieren su resultado. Como ejemplo, supongamos el siguiente par de reglas de un SBR para clasificación de animales, en las que se clasifica un animal en vertebrado o invertebrado, en función de si tiene esqueleto o no:

```
R1:  SI (animal ?a) ∧ (esqueleto ?a sí) ENTONCES (vertebrado ?a)
R1:  SI (animal ?a) ∧ (esqueleto ?a no) ENTONCES (invertebrado ?a)
```

Durante la generación de la red RETE, el compilador agrupa los patrones *animal* y *esqueleto* (véase Figura 3.16) y separa los patrones *sí* y *no*. De esta forma, durante la ejecución los patrones agrupados se equipararán una sola vez; mientras que la separación en ramas distintas de los patrones *sí* y *no* permitirá la equiparación de ambos de forma individual. Al final de la red, la unión de ambas ramas permitirá la equiparación global de la regla, además de verificar si la variable *?a* se equipara siempre con la misma ocurrencia, ya que dicha variable aparece más de una vez en una misma regla.

2. En cada ciclo los cambios en la BH suelen ser muy pocos, aunque los contenidos sean muchos. Por tanto, es muy ineficiente equiparar, en cada ciclo, toda la BH. La solución propuesta por el algoritmo RETE consiste en guardar el resultado de la equiparación durante un ciclo, para que pueda ser usado nuevamente en el ciclo siguiente. De esta manera, el coste computacional depende de la velocidad de cambio de la BH, que suele ser baja, y no de su tamaño, que suele ser grande. RETE opera de la siguiente manera:
 - (a) Guarda las instanciaciones, resultado de las equiparaciones, de ciclo a ciclo (tanto de las equiparaciones completas de las condiciones de las reglas como de las parciales).
 - (b) Actualiza esta información con los cambios que se producen en la BH después de la ejecución de cada ciclo. Para cada nuevo dato en la BH, obtiene las nuevas instanciaciones que se generan y las añade al conjunto de las

instanciaciones resultado de previos ciclos. Para cada dato que se borra de la BH, RETE elimina de dicho conjunto las instanciaciones a las que dio lugar en ciclos previos.

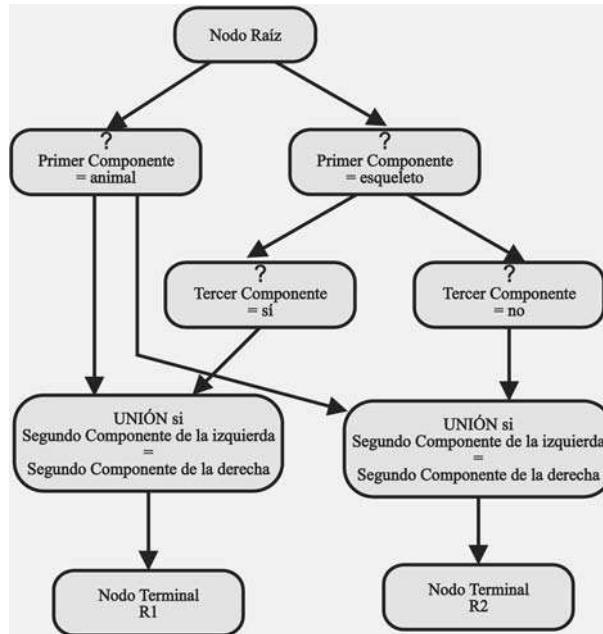


Figura 3.16: Red RETE para las reglas 1 y 2 del ejemplo de clasificación de animales.

Para aplicar el algoritmo RETE, es necesario modificar el ciclo de reconocimiento-acción (véase Figura 3.17). A partir del conjunto conflicto, la etapa de resolución selecciona la regla a aplicar. La etapa de acción ejecuta las acciones especificadas en la parte derecha de la regla seleccionada. Estas acciones suelen producir cambios en la BH. Estos cambios se notifican a la red RETE, para que ésta los equipe con las condiciones de las reglas y genere los cambios oportunos en el conjunto conflicto.

La red RETE puede considerarse como una función que traslada los cambios de la BH a cambios en el conjunto conflicto. Cada vez que se producen cambios en la BH se crean estructuras, llamadas señales o testigos, que describen dichos cambios. Un *testigo* es un par de elementos, donde el primero de ellos puede tomar un valor en el conjunto $\{+, -\}$, y el segundo es una lista de elementos que representan los cambios de la BH. El signo $+$ indica que los elementos han sido añadidos a la BH; mientras que el signo $-$ que han sido eliminados de la BH. Todos los nodos de la red, excepto el nodo raíz y el terminal, tienen asociada una memoria, donde se almacenan los testigos. Retomemos el ejemplo del sistema de clasificación de animales (véase la Figura 3.16). Supongamos que el nodo raíz recibe el testigo $+(animal\ Pancho)$, lo que significa que es un elemento que se ha añadido a la BH. El nodo raíz propaga el testigo al nodo

hijo izquierdo. El testigo pasa el control de dicho nodo hijo, ya que su primer componente es *animal*. A continuación, se propaga a los hijos izquierdo y derecho (nodos unión), y queda almacenado en la memoria de dichos nodos unión (véase Figura 3.18) por tratarse de un testigo positivo. En el caso de que se hubiese recibido un testigo negativo, suprimiría de la memoria un testigo positivo almacenado previamente. Si no hubiese nada en la memoria, entonces el testigo negativo desaparecería.

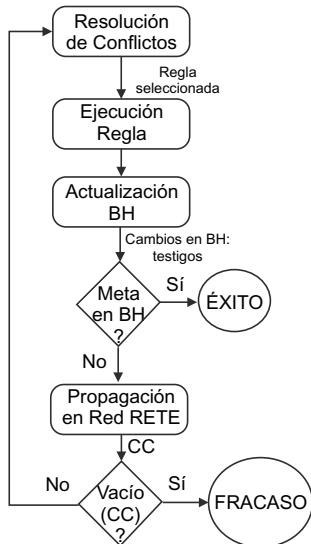


Figura 3.17: Ciclo de reconocimiento-acción del algoritmo RETE (CC es el conjunto conflicto).

Si a posteriori se recibe el testigo *+(esqueleto Pancho sí)*, éste pasaría el control al nodo que está a la derecha del nodo raíz. A continuación, se propaga al hijo izquierdo (nodo unión izquierdo), y se almacena en la memoria de dicho nodo unión. En este momento, ya están los dos elementos necesarios para que se ejecute el nodo unión izquierdo. El nodo terminal de la regla R1 informa de los cambios, es decir, de la generación de la instancia *{+ R1 (?a Pancho)}* (véase la Figura 3.19).

3.5 Técnicas de resolución de conflictos

El método de resolución de conflictos selecciona, a partir del conjunto conflicto, la instancia a aplicar. La forma de realizar esta selección es importante, ya que de ello depende tanto el tiempo de respuesta del sistema ante cambios del entorno como la facultad de ejecutar secuencias de acciones relativamente largas. Las principales técnicas de resolución de conflictos que más se han usado son las siguientes:

1. *Seleccionar la primera regla que se equipara con los contenidos de la BH.* En este caso el coste de control es pequeño, ya que la selección de la regla no implica

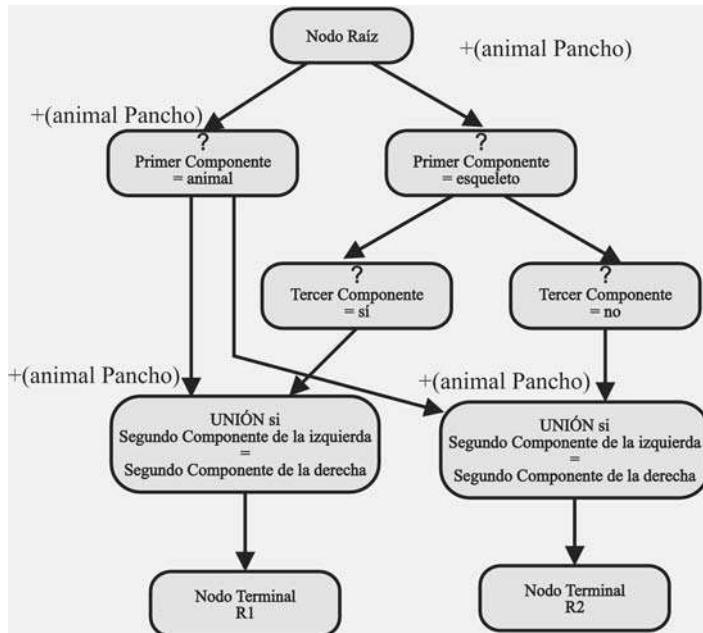


Figura 3.18: Red RETE para las reglas 1 y 2 del ejemplo de clasificación de animales, recibiendo el testigo $+(animal\ Pancho)$.

cálculos costosos. Sin embargo, conlleva un coste de aplicación de las reglas elevado, ya que es necesario ensayar un gran número de reglas para encontrar la solución buscada. A veces, esta técnica se combina con otras que consisten en ordenar la BC, tales como:

- Ordenación de las reglas en la BC*, colocando en primer lugar las que se deben examinar antes. Este método es sencillo, pero dificulta el mantenimiento de BC grandes.
 - Ordenación de las condiciones dentro de cada regla*. Esta técnica es muy usada en los SBR con encadenamiento hacia atrás. Durante la ejecución del sistema, si no se verifica una condición, ya no se necesita investigar el resto de las condiciones que aparecen a continuación dentro de la misma regla. Para que este método sea eficaz, durante la definición de las reglas, hemos de ser cuidadosos a la hora de colocar primero las condiciones que tienen menos probabilidades de verificarse.
2. *Seleccionar la regla de prioridad más alta*. La prioridad se definirá durante la construcción del SBR y será función de las necesidades y características del problema.

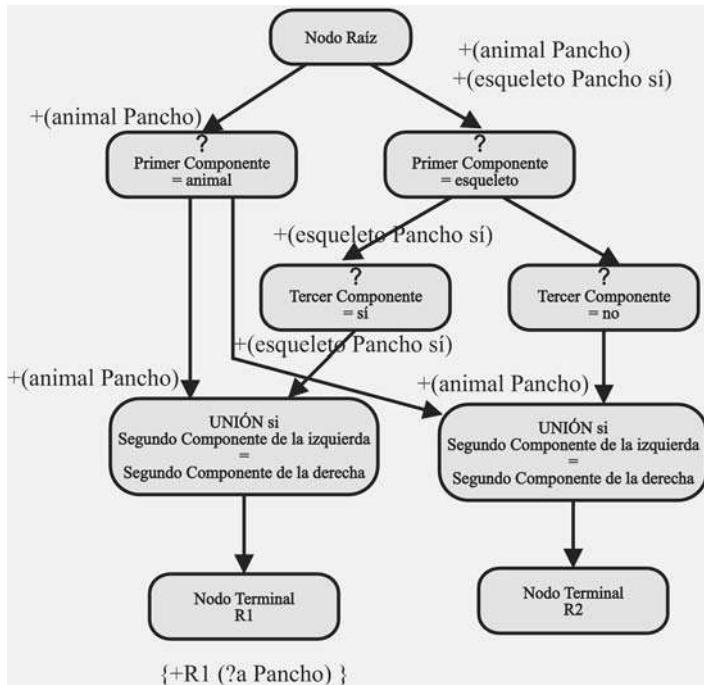


Figura 3.19: Red RETE para las reglas 1 y 2 del ejemplo de clasificación de animales, recibiendo los testigos $+ (\text{animal Pancho})$ y $+ (\text{esqueleto Pancho sí})$.

3. *Seleccionar las instanciaciones más específicas.* En este caso, se supone que las instancias más específicas se adaptan mejor a la situación planteada.
4. *Seleccionar arbitrariamente una regla,* dentro de un conjunto de reglas con igual posibilidad de ser efectivas.
5. *Seleccionar las instanciaciones que tienen elementos más recientemente añadidos a la BH.* Para ello, la BH debe poseer un contador que indique el ciclo que se está ejecutando en cada momento. Además, a cada elemento de la BH se le añade un número, que indica su antigüedad en la BH. Cada vez que se añade o modifica un elemento en la BH, se le asigna a dicho elemento el valor actual del contador. Así, los elementos más recientes serán aquellos que posean un número más elevado. Por ejemplo, supongamos una BC con las siguientes 4 reglas simbólicas:

Regla R1: $a?x \wedge ?xc \rightarrow \dots$

Regla R2: $ac \wedge bc \rightarrow \dots$

Regla R3: $bc \wedge ?xb \wedge cd \rightarrow \dots$

Regla R4: $a?x \wedge \neg ab \wedge bc \rightarrow \dots$

y supongamos que en el ciclo actual, la BH contiene los siguientes hechos:

BH: (98 bc),(91 ab),(94 cd)

La equiparación de la BC con la BH resulta en el siguiente conjunto conflicto:

Conjunto conflicto: (R1, (91 ab) \wedge (98 bc)), (R3, (98 bc) \wedge (91 ab) \wedge (94 cd))

en el que el elemento más reciente de las dos instanciaciones es *bc* y, en ambos casos, tienen la misma antigüedad. Comparando el segundo elemento más reciente en ambas instanciaciones, vemos que lo posee el elemento *cd* de la regla R3. Por tanto, ésta será la instancia seleccionada.

Si consideramos ahora otra BH, tal como:

BH: (47 ac),(90 bc)

obtenemos el siguiente conjunto de conflicto:

Conjunto conflicto: (R2, (47 ac) \wedge (90 bc)), (R4, (47 ac) \wedge (90 bc))

las dos instanciaciones poseen elementos con la misma antigüedad. Por lo tanto, en este caso, este método no permite seleccionar una.

6. *Seleccionar una instancia que no se haya ejecutado previamente.* Con esta técnica se evita la ejecución de la misma instancia indefinidamente. Esta técnica se conoce como *principio de refracción*. Dos instanciaciones se consideran diferentes si, correspondiendo a la misma regla y a los mismos elementos de la BH, una de ellas está incluida en el conjunto conflicto durante unos determinados ciclos, posteriormente desaparece del conjunto conflicto y, por último, aparece nuevamente en el conjunto conflicto. Supongamos el siguiente ejemplo:

Regla R1:	$a \rightarrow \text{Añadir}(b)$
Regla R2:	$b \wedge \neg c \wedge \neg d \rightarrow \text{Añadir}(c)$
Regla R3:	$e \rightarrow \text{Añadir}(d)$
Regla R4:	$d \rightarrow \text{FIN}$
Regla R5:	$a \wedge c \rightarrow \text{Añadir}(e) \wedge \text{Borrar}(c)$
BH:	(1 a)

- Primer ciclo:

Conjunto conflicto: (R1, (1 a)).

Se aplica la única instancia del conjunto conflicto:

BH: (1 a),(2 b)

- Segundo ciclo:

Conjunto conflicto: (R1, (1 a)),(R2, (2 b)).

Como la instancia de R1 no ha desaparecido desde su creación y ya se ha aplicado, se ejecutará la de R2:

BH: (1 a),(2 b),(3 c)

- Tercer ciclo:

Conjunto conflicto: (R1, (1 a)),(R5, (1 a) \wedge (3 c)).

Como la instancia de R1 no ha desaparecido desde su creación y ya se ha aplicado, se ejecutará la instancia de R5:

BH: (1 a),(2 b),(4 e)

- Cuarto ciclo:

Conjunto conflicto: (R1, (1 a)),(R2, (2 b)),(R3, (4 e)).

La instancia de R1 no puede aplicarse, por lo dicho anteriormente. Sin embargo, la instancia de R2 puede aplicarse porque se acaba de generar. En este caso, habrá que decidir entre aplicar (R2, (2 b)) o (R3, (4 e)). Si suponemos que se selecciona la de menor índice, se ejecutará la instancia de R2:

BH: (1 a),(2 b),(4 e),(5 c)

- Quinto ciclo:

Conjunto conflicto:(R1, (1 a)),(R3, (4 e)),(R5, (4 e) \wedge (5 c))

La instancia de R1 no puede aplicarse, por lo dicho anteriormente. Sin embargo, la instancia de R3 sí se puede aplicar porque no se aplicó desde que se generó. En este caso, habrá que decidir entre aplicar (R3, (4 e)) o (R5, (4 e) \wedge (5 c)). Por el criterio que aplicamos antes, se ejecutará la instancia de R3:

BH: (1 a),(2 b),(4 e),(5 c),(6 d)

- Sexto ciclo:

Conjunto conflicto: (R1, (1 a)),(R3, (4 e)),(R4, (6 d)),(R5, (4 e) \wedge (5 c)).

Las instancias de R1 y R3 no pueden aplicarse. En este caso, habrá que decidir entre aplicar (R4, (6 d)) o (R5, (4 e) \wedge (5 c)). Por el criterio que aplicamos antes, se ejecutará la instancia de R4 y se terminará la ejecución.

Todos los métodos comentados no se suelen usar aisladamente, sino combinados. Para ello, se establece un orden de prioridad en los métodos: se aplica el primer método escogido al conjunto conflicto, y sobre el subconjunto de instanciaciones obtenido se aplica el segundo método, y así sucesivamente. Cuando se asigna la misma prioridad a dos métodos, ambos se aplican al mismo conjunto de instanciaciones y se realiza la intersección de los dos conjuntos obtenidos. Por último, en aplicaciones reales, tales como la medicina, puede llegar a ser más operativo estudiar exhaustivamente las posibles consecuencias de la aplicación de todas las posibles soluciones, y en función de los resultados de estas simulaciones escoger la más adecuada, en lugar de aplicar un criterio establecido a priori.

3.6 Ventajas e inconvenientes

Una de las características más importantes que presentan las reglas es la independencia entre ellas. Las reglas sólo se comunican a través de la BH: una regla no puede disparar directamente la ejecución de otra regla, es decir, el consecuente de una regla no puede contener otra regla. Por tanto, la BC está formada por un conjunto de reglas independientes entre sí. Esto proporciona una gran modularidad a los SBR, en donde el conocimiento puede introducirse y modificarse en cualquier orden. Esto, a su vez, facilita el mantenimiento del sistema, ya que es muy fácil introducir modificaciones en el sistema. Esto contrasta con los sistemas de programación convencional, en lo que cualquier modificación es difícil de efectuar. Por ejemplo, en un SBR para finanzas, un grupo de reglas pueden hacer deducciones sobre cómo varía la inflación y otro grupo puede hacer conclusiones sobre cómo varían los tipos de intereses. Si se quieren añadir nuevas reglas para hacer conclusiones sobre flujos de inversiones extranjeras, basta con añadir dichas reglas de forma incremental. Esto no afectará necesariamente a las reglas que ya estaban en el sistema. Sin embargo, en SBR con grandes BC, la modificación de reglas (adición o borrado) puede originar efectos colaterales inesperados sobre las otras reglas almacenadas previamente, ya que el único punto de unión entre éstas es la BH. En la práctica, se hace necesario buscar métodos para estructurar la BC, que faciliten la depuración y modificación del SBR.

Otra diferencia importante entre los SBR y la programación convencional basada en procedimientos es la forma de selección de una condición entre varias que se satisfacen simultáneamente. En la aproximación basada en procedimientos, se selecciona la siguiente condición en una secuencia lineal de condicionales (cuyo orden ha sido establecido), mientras que en un SBR el mecanismo de control tiene en cuenta diferentes factores para tomar la decisión, y ésta se realiza en el momento de la resolución.

Por otra parte, la representación declarativa del conocimiento es más cercana al pensamiento humano. De hecho, cuando un experto trata de transmitir su experiencia, con frecuencia expresa su conocimiento en forma de reglas declarativas: *en la situación S se aplica X o la hipótesis H se verifica cuando las premisas P se verifican*. Además, las reglas son fáciles de leer y entender, lo que supone una gran ventaja para su uso por gente que no es especialista en informática. Además, un SBR puede seguir el rastro fácilmente al conjunto de reglas que aplicó durante el proceso de inferencias.

Esto confiere a los SBR una propiedad que no poseen los sistemas de programación convencionales, que es el poder de autoexplicación.

Sin embargo, el uso de los SBR también posee inconvenientes. Por ejemplo, los SBR puros no permiten usar estructuras de control usuales en los sistemas de programación convencionales, tales como condicionales, iteraciones, llamadas a funciones, recursiones, etc. No obstante, la implementación de ciertas operaciones perfectamente definidas es más sencilla usando este tipo de estructuras. Por ello, las herramientas actuales permiten en alguna medida el uso de funciones escritas en algún lenguaje de programación.

El diseño de SBR no es trivial debido a la dificultad en el modelado del conocimiento. El tipo de granularidad escogido influye en su eficiencia: una granularidad alta puede no captar los conceptos básicos del problema y una granularidad excesivamente fina aumenta considerablemente el número de reglas del sistema y conlleva una pérdida de generalidad entre éstas, que obliga a una frecuente actualización del sistema.

En general, las ventajas e inconvenientes de los SBR que hemos comentado hasta el momento se derivan de la representación declarativa del conocimiento y, por tanto, están presentes en las otras técnicas de representación del conocimiento. Pasamos ahora a comentar las diferencias fundamentales de los SBR con la lógica y la representación mediante marcos. En su formulación más simple, las reglas pueden considerarse como una versión reducida de la lógica de predicados. La diferencia fundamental entre ambos métodos es que las reglas reducen la capacidad de representación (potencia expresiva) y la capacidad de inferencia de la lógica de predicados con el fin de obtener una mayor eficiencia. En cuanto a la expresividad, las reglas que hemos estudiado en este tema contienen variables y cuantificadores universales implícitos, pero en general no permiten representar cuantificadores existenciales, por lo que ofrecen mayor expresividad que la lógica de proposiciones pero no llegan a tener la expresividad de la lógica de predicados. La reducción en la capacidad de inferencia de los SBR en comparación con la lógica de predicados aparece en la imposibilidad de aplicar el modus tollens (un SBR sólo puede establecer el consecuente cuando se ha afirmado el antecedente).

Otra ventaja de los SBR con respecto a la lógica, además de la eficiencia, es la posibilidad de tratamiento de la incertidumbre. Los SBR suelen manipular datos obtenidos de la experiencia, que pueden ser conocidos sólo de forma aproximada. Es decir, no reflejan implicaciones lógicas sino, más bien, las convicciones de un experto. Una forma de tratar esta incertidumbre es añadir a cada regla un factor de certeza, que refleja el grado de confianza que el experto le concede a cada una de las reglas. Además, aunque el SBR parte de datos ciertos, las conclusiones a las que puede llegar pueden no ser seguras. Por ejemplo, la regla

SI un animal vuela y pone huevos **ENTONCES** es un pájaro

no expresa una inferencia cierta, ya que los insectos son animales que vuelan y ponen huevos, pero no son pájaros. Sin embargo, necesitamos trabajar con este tipo de reglas

de forma válida para evitar deducir resultados sin valor. Se han propuesto diferentes modelos de razonamiento aproximado que permiten gestionar la incertidumbre: cálculo de probabilidades, de plausibilidad, de credibilidad, modal, no monótonos y difuso. Dichos modelos se revisan en los capítulos 2, 6 y 7.

Resumiendo, las representaciones lógicas son puramente sintácticas. Sus reglas de inferencia son procedimientos estrictamente sintácticos que operan sobre fórmulas bien formadas, a pesar de lo que éstas representen. Los SBR son también principalmente sintácticos, ya que usan únicamente información sintáctica para decidir qué reglas desestiman. Sin embargo, existen sistemas que tienen más semántica englobada en ellos. Por ejemplo, los sistemas que proporcionan un soporte explícito para el tratamiento de la incertidumbre, el mecanismo de inferencias usa las semánticas de dichos elementos para guiar la inferencia.

Por último, comparando la representación mediante marcos y los SBR, la principal ventaja de los primeros es que representan el conocimiento de forma estructurada (véase el capítulo 4). Toda la representación mediante un objeto se reúne en un marco, en vez de estar dispersa en un conjunto de reglas sin ninguna o poca estructura. Esto proporciona más eficiencia y mayor facilidad de mantenimiento de la BC. Sin embargo, los marcos suelen resultar más pobres que las reglas cuando se trata de representar afirmaciones de la forma *SI x, y ∧ z ENTONCES w*. Las representaciones mediante marcos son normalmente más semánticas, es decir, sus procedimientos para el razonamiento son más variados, más eficaces y están más estrechamente relacionados con los tipos específicos de conocimiento. En contraste, es difícil expresar afirmaciones más complejas que la herencia de propiedades.

3.7 Dominios de aplicación

Hay dominios en los cuales los problemas se modelan de forma sencilla mediante los SBR, mientras que en otros dominios no sucede esto. En esta sección, vamos a repasar brevemente los dominios que son adecuados para los SBR, tratando de buscar una explicación a ello. En primer lugar, podemos distinguir dos clases diferentes de problemas: aquellos que se pueden modelar como un conjunto de múltiples estados y aquellos que vienen descritos por medio de una teoría concisa y unificada. Un ejemplo de los primeros es la medicina clínica, en donde hay muchos estados asociados a las acciones que se pueden llevar a cabo. Esto suele deberse a la falta de una teoría concisa o a la complejidad del sistema a modelar. En estos casos, es sencillo asociar múltiples reglas modulares a los diferentes estados independientes. Ejemplos del segundo tipo de problema son las áreas de la física o las matemáticas, en las que unos pocos principios contienen mucho del conocimiento requerido.

Teniendo en cuenta la complejidad del flujo de control, se puede distinguir entre problemas representados por un conjunto de acciones independientes (por ejemplo, tareas tales como la monitorización) y problemas representados por una colección compleja de múltiples procesos paralelos con varios subprocesos dependientes. En el primer caso, la comunicación entre las acciones es muy limitada y, como vimos, ésta es una característica importante de los SBR. Por tanto, el problema se puede modelar

fácilmente mediante un SBR. Sin embargo, los problemas con flujos de control (búcles y saltos) complicados requieren comunicación explícita entre las acciones (ya que unas acciones invocan otras). Los SBR no proporcionan mecanismos para establecer estas comunicaciones. Además, existen dominios en los que el conocimiento se puede separar claramente de la forma en que se usa. Por ejemplo, estableciendo hechos sin asumir cómo se van a usar dichos hechos. Un ejemplo concreto son los problemas de taxonomía propios de la biología. Estos dominios son adecuados para modelarlos mediante un SBR. Alternativamente, podemos escribir descripciones de procedimientos que indiquen la forma de alcanzar un objetivo. En estos casos, es muy difícil especificar exactamente de antemano cómo va a ser usado un hecho dado, mediante un SBR.

Resumiendo, los SBR se muestran adecuados para modelar dominios donde las tareas se caracterizan por el reconocimiento de un número elevado de estados distintos. En los sistemas basados en procedimientos es difícil organizar y chequear un gran número de variables de estado y sus correspondientes transferencias de control. Sin embargo, esta tarea es más sencilla con los SBR, donde cada regla se puede ver como un proceso *demonio* que espera la ocurrencia de un estado específico. Además, toda regla puede dispararse en cualquier momento de la computación. Su disparo sólo depende del estado de la BH al final de cada ciclo.

3.8 Resumen

Los SBR permiten modelar gran cantidad de conocimiento útil que se suele expresar mediante reglas sencillas del tipo SI ... ENTOCES. Se debe a que utilizan el razonamiento deductivo para llegar a obtener conclusiones lógicas. Como hemos visto, los SBR pueden interpretar estas reglas de dos formas. Una primera alternativa consiste en aplicar las reglas de la forma condición-acción en un control con encadenamiento hacia delante. La segunda opción considera las reglas como conjuntos de implicaciones lógicas, a partir de las cuales se obtienen las deducciones en un control con encadenamiento hacia atrás. En ambos casos, se requieren técnicas de equiparación entre el estado actual de la BH y las condiciones de las reglas para determinar, entre todas las reglas, cuáles de ellas se pueden aplicar en un ciclo dado de la inferencia. El algoritmo de equiparación RETE evita examinar todas las reglas de la BC con todos los datos de la BH. Para ello, genera una red que agrupa las condiciones iguales de las reglas en un único nodo y equipara, en cada ciclo, sólo las modificaciones más recientes de la BH, almacenando dichas modificaciones de ciclo en ciclo. Para un estudio más profundo sobre el algoritmo RETE, se aconseja revisar el tema dedicado a los SBR de [Borrajo y otros, 1993; Winston, 1994]. El resultado del proceso de equiparación es una lista de reglas cuyas partes izquierdas se han equiparado con la descripción del estado actual. Los métodos de resolución de conflictos permiten determinar, entre varias reglas seleccionadas, cuál de ellas se debe aplicar.

En las dos siguientes secciones proponemos algunos ejercicios sobre SBR. El lector interesado puede encontrar también un gran número de problemas resueltos en [Fernández y otros, 1998].

3.9 Ejercicios resueltos

3.1. Se desea diseñar un sistema de ayuda a la toma de decisiones en la realización de cesáreas a mujeres embarazadas en período de parto. Se requiere modelar el siguiente conocimiento: *Si la mujer embarazada tiene el bebé en posición podálica, entonces se recomienda realizar cesárea.*

De las cuatro reglas que se presentan a continuación, ¿cuáles de ellas son correctas? ¿Por qué?

- **R1:** $(\text{bebé } ?b) \wedge (\text{posición } ?b \text{ Podálica}) \rightarrow (\text{cesárea } ?b)$.

Solución: Veamos primeramente el significado de cada condición del antecedente:

- $(\text{bebé } ?b)$ expresa el bebé dado por la variable $?b$.
- $(\text{posición } ?b \text{ Podálica})$ indica que la posición del bebé $?b$ es Podálica.

En el consecuente de la regla aparece la misma variable que en las dos condiciones del antecedente. Supongamos que durante la etapa de equiparación, la variable $?b$ se queda ligada al valor *bebéJuan*, y que la regla R1 forma parte del conjunto conflicto, entonces la instanciación que se generaría sería:

$$\{(bebé ?b) \wedge (\text{posición } ?b \text{ Podálica}) \rightarrow (\text{cesárea } ?b), (?b \text{ bebéJuan})\}$$

Dicha instanciación expresa que debe realizarse la cesárea al *bebéJuan*, lo cual es incorrecto, ya que una cesárea se realiza a una madre en parto, no a un bebé.

■

- **R2:** $(\text{bebé } ?b) \wedge (\text{posición } ?b \text{ Podálica}) \rightarrow (\text{cesárea } ?m)$.

Solución: En el consecuente de esta regla aparece la variable $?m$, la cual no está en el antecedente. Por tanto, en ninguna instanciación de la regla se establecerá una ligadura a dicha variable. Es decir, durante la ejecución de la regla, la variable no tendrá asignado ningún valor. Por tanto, la regla está incorrectamente diseñada.

■

- **R3:** $(\text{bebé } ?b) \wedge (\text{posición } ?b \text{ Podálica}) \wedge (\text{madre } ?m) \rightarrow (\text{cesárea } ?m)$

Solución: Veamos primeramente el significado de cada condición del antecedente:

- $(\text{bebé } ?b)$ expresa el bebé dado por la variable $?b$.
- $(\text{posición } ?b \text{ Podálica})$ indica que la posición del bebé $?b$ es Podálica.
- $(\text{madre } ?m)$ representa la madre $?m$.

En el consecuente de la regla aparece la misma variable $?m$ que en la última condición del antecedente. Supongamos que la BH está formada por dos madres, *María* y *Marta*, cada una de ellas embarazadas de un bebé, *bebéJuan*, en posición *Podálica*, y *bebéCarlos*, en posición normal. Durante la etapa de equiparación, se generaría dos instanciaciones:

1. {R3, ($?b$ bebéJuan), ($?m$ María)}
2. {R3, ($?b$ bebéJuan), ($?m$ Marta)}

Sin embargo, la segunda instancia es incorrecta, ya que expresa la necesidad de realizar la cesárea a la madre *Marta*, cuyo *bebéCarlos* está en posición normal. Ello es debido a que la regla omite la relación entre madre e hijo.

■

- **R4:** (bebé $?b$) \wedge (posición $?b$ Podálica) \wedge (madre $?m$) \wedge (madre-de $?b$ $?m$) \rightarrow (cesárea $?m$)

Solución: Esta regla resuelve el problema anterior, mediante la incorporación de una condición que expresa la relación entre madre e hijo:

$$\text{(madre-de } ?b ?m\text{)}$$

■

3.2. Considérese ahora la regla R4 del ejercicio anterior y la siguiente BH, correspondiente a un embarazo gemelar, con uno de los bebés en posición podálica:

$$\text{BH}=\{\text{(madre María)} \text{ (madre-de bebéJuan María)} \text{ (madre-de bebéJacobo María)} \\ \text{ (posición bebéJuan Podálica)} \text{ (bebé bebéJuan)} \text{ (bebé bebéJacobo)}\}$$

¿Se recomienda realizar la cesárea a la madre?

Solución: Durante la etapa de equiparación, se generaría sólo una instancia:

$$\{\text{R4, (?b bebéJuan), (?m María)}\}$$

con lo que se ejecutaría la regla y se recomendaría realizar la cesárea a *María* por estar uno de sus dos bebés en posición podálica.

■

3.3. Supóngase una empresa de mensajería que transporta paquetes y cartas. En la Tabla 3.3 se presentan las tarifas base que se aplican al enviar cartas y paquetes menores de 2 Kg entre una ciudad origen y una destino. Las tarifas base son para entregas al día siguiente de la recepción del paquete en la empresa.

Si la entrega se tiene que realizar en el mismo día que la recogida, se aplicará un suplemento tanto para las cartas como para los paquetes de 6 euros. Si el paquete pesa más de 2 kilos, el suplemento es de 1 euro por cada 100 gramos de más. Se pide:

ORIGEN	DESTINO	CARTA	PAQUETE
Madrid	Barcelona	7	12
Madrid	Toledo	3	8
Madrid	Badajoz	5	10
Barcelona	Cádiz	10	16
Barcelona	Gerona	3	8
Barcelona	Santiago	8	15

Tabla 3.3: Tarifas base de los costes de envío (en euros).

1. Teniendo en cuenta los contenidos iniciales de la BH, represéntela y describa dichos contenidos iniciales.

Solución: Inicialmente, la BH contendrá los 12 hechos descritos en la Tabla 3.3. Como la BH contiene instanciaciones de las condiciones de las reglas, los hechos siguen la misma sintaxis que las reglas (véase sección 3.4.1): ir encerrados entre paréntesis y empezar por una constante. Como necesitamos representar los costes de envíos, podemos llamar a dicha constante *coste*. Así, la sintaxis de las instanciaciones tendría un formato como:

(coste)

Además, para representar la información de la Tabla 3.3, los hechos deben tener información sobre el origen ($?o$) y el destino ($?d$) del envío, así como el tipo ($?t$) de envío (si es carta o paquete) y el precio ($?p$). Por tanto, la sintaxis completa de las instanciaciones sería:

(coste $?o$ $?d$ $?t$ $?p$)

Teniendo en cuenta dicha sintaxis, la BH inicial sería:

BH={ (coste Madrid Barcelona Carta 7) (coste Madrid Barcelona Paquete 12)
(coste Madrid Toledo Carta 3) (coste Madrid Toledo Paquete 8)
(coste Madrid Badajoz Carta 5) (coste Madrid Badajoz Paquete 10)
(coste Barcelona Cádiz Carta 10) (coste Barcelona Cádiz Paquete 16)
(coste Barcelona Gerona Carta 3) (coste Barcelona Gerona Paquete 8)
(coste Barcelona Santiago Carta 8) (coste Barcelona Santiago Paquete 15)}

■

2. Construir la base de reglas que formalice dichos conocimientos utilizando el menor número de reglas posibles.

Solución: Primeramente, necesitaremos establecer la representación completa de la BH, indicando qué variables se requieren manejar.

- (a) Envío: $?e$.
- (b) Origen del envío: $?o$.
- (c) Destino del envío: $?d$.
- (d) Tipo de envío (carta o paquete): $?t$.
- (e) Precio del envío: $?p$.
- (f) Peso del envío: $?k$.
- (g) Urgencia del envío (el mismo día o al día siguiente): $?u$.

A continuación, describimos las reglas en lenguaje natural:

- (a) R1: Dada la BH de partida planteada, necesitamos una regla que inicialmente asigne la tarifa base a cada envío. Por ejemplo, supongamos el envío $e1$ de una carta de Madrid a Barcelona a entregar el mismo día. Esta regla asignaría la tarifa base a dicho envío $e1$, es decir, añadiría a la BH el precio de 7 euros para el envío $e1$.
- (b) R2: Si en la BH existe un hecho para $e1$ que almacena un precio, se necesitará una regla que incremente el precio del envío con el suplemento adecuado, si el peso del envío es mayor que 2 Kg. Dicha regla sólo se podrá disparar una vez por envío, ya que el incremento sólo se aplica una vez.
- (c) R3: Si en la BH existe un hecho para $e1$ que contiene un precio, se necesitará otra regla que incremente el precio del envío con el suplemento adecuado, si el envío es urgente. Esta regla también se debe disparar una sola vez por envío. En el ejemplo, la regla R3 se dispararía para incrementar el valor del precio de $e1$ almacenado en la BH en 6 euros.

Seguidamente, identificamos los elementos de condición y acción que aparecerán en cada regla:

- (a) La regla R1 necesita equiparse con alguno de los hechos de partida de la BH (descritos en la parte 1 del ejercicio) con el fin de obtener el precio base. Es decir, requiere una condición del tipo:

$$(\text{coste } ?o \ ?d \ ?t \ ?p)$$

Además debe equiparar el envío concreto que estará también en la BH. Para representar el envío tenemos dos estrategias. La primera consiste en especificar en una sola condición toda la información sobre el envío. Esta estrategia daría lugar a la siguiente condición:

$$(\text{envío } ?e \ ?o \ ?d \ ?t \ ?k \ ?u)$$

Esta opción reduce el tiempo de equiparación entre la BC y la BH, ya que existen menos condiciones a equiparar. Sin embargo, para llegar a obtener

resultados de la equiparación, la BH requiere disponer de toda la información inicialmente. Por ejemplo, para el envío e_1 , no se requiere conocer exactamente el peso, ya que no supera los 2 Kg. Si la BH no tiene constancia del peso de e_1 , una regla con la condición (*envío ?e ?o ?d ?t ?k ?u*) nunca llegará a equiparse con el envío e_1 de la BH y, por tanto, no se obtendrá ningún resultado. Para resolver este problema, una segunda estrategia consiste en particionar las condiciones, tal como:

```
(envío ?e)
(origen ?e ?o)
(destino ?e ?o)
(tipo ?e ?t)
(peso ?e ?k)
```

Sin embargo, una partición excesiva de las condiciones de las reglas, puede llegar a aumentar de manera significativa el tiempo de la equiparación. Como consecuencia de ello, la estrategia ideal es una solución de compromiso entre ambas alternativas: particionar las condiciones, de manera que permitan equiparaciones exitosas de la BC con la BH cuando no se conocen datos que no se requieren, pero no demasiado para evitar incrementar de manera excesiva e innecesaria el tiempo de equiparación.

En el ejemplo, la información mínima que identifica un envío es el origen, el destino y el tipo. Esto da lugar a las siguientes condiciones:

```
(envío ?e ?o ?d ?t)
(precio ?e ?p)
(peso ?e ?k)
(urgente ?e ?u)
```

Teniendo esto en cuenta, la regla R1 podría expresarse como:

SI (*envío ?e ?o ?d ?t*) \wedge (*coste ?o ?d ?t ?p*) **ENTONCES** Añadir(*precio ?e ?p*)

- (b) La regla R2 requiere que la BH contenga el precio del envío, para incrementarlo con el suplemento, siempre que el peso sea mayor que 2Kg. Por tanto, se requerirán tres cláusulas:

- i. La condición que chequea si existe en la BH un precio para dicho envío:

```
(precio ?e ?p)
```

Esta cláusula, además, liga la variable $?p$ con el precio actual, con el fin de incrementarlo en la parte de acción de la regla con el suplemento correspondiente.

ii. La condición que obtiene el peso del envío para el que ya existe un precio:

(peso ?e ?k)

Esta condición es necesaria para poder chequear el peso con la siguiente.

iii. La condición que chequea el peso:

(test ($>?k$ 2))

Teniendo en cuenta estas cláusulas, la regla R2 podría ser:

SI (precio ?e ?p) \wedge (peso ?e ?k) \wedge (test ($>?k$ 2))
ENTONCES
Añadir(precio ?e ?p+(?k-2)*10)

Veámos ahora qué sucedería si el envío e1 pesase 2.2 Kg. Inicialmente, la BH almacenaría los contenidos de la Tabla 3.3 más la información inicial sobre el envío e1:

(envío e1 Madrid Barcelona carta), (peso e1 2.2), (urgente e1 sí)

En el primer ciclo, el conjunto conflicto estaría únicamente formado por la instanciación de la regla R1:

(R1, (envío e1 Madrid Barcelona carta) (coste e1 Madrid Barcelona 7))

Por lo que se seleccionaría dicha instanciación y se ejecutaría, añadiendo a la BH la siguiente información:

(precio e1 7)

En el ciclo 2, el conjunto conflicto estaría formado por las siguientes instanciaciones:

{(R1, (envío e1 Madrid Barcelona carta) (coste e1 Madrid Barcelona 7)),
(R2, (precio e1 7) (peso ?e1 2.2))}

Si consideramos el principio de refracción (véase sección 3.5), la instanciación R1 no podría dispararse nuevamente, por lo que se seleccionaría la instanciación de la regla R2, que se ejecutaría y añadiría a la BH la siguiente información:

(precio e1 9)

Con lo cual, la BH contendría la Tabla 3.3 más la siguiente información:

(envío e1 Madrid Barcelona carta), (peso e1 2.2),
(urgente e1 sí), (precio e1 7), (precio e1 9)

Sin embargo, el precio de un envío debería de ser único. Por tanto, la regla R2 está incorrectamente diseñada. Una forma de solucionar el problema, es incrementar una cláusula en la parte de acción de la regla, que borre el precio del envío. Dicha cláusula eliminaría el primer precio de e1 que se encontrase al equiparar la BH; en el ejemplo, (precio e1 7). La regla R2 sería ahora:

SI (precio ?e ?p) \wedge (peso ?e ?k) \wedge (test ($>?k$ 2))
ENTONCES
Añadir(precio ?e ?p+(?k-2)*10) Borrar (precio ?e ?p)

Teniendo en cuenta este segundo diseño para la regla R2, una vez que se ejecuta dicha regla en el ciclo 2, la BH contendría la información de la Tabla 3.3 más la siguiente información:

(envío e1 Madrid Barcelona carta), (peso e1 2.2), (urgente e1 sí),
(precio e1 9)

En el ciclo 3, el conjunto conflicto estaría formado por las siguientes instancias:

{(R1, (envío e1 Madrid Barcelona carta) (coste e1 Madrid Barcelona 7)),
(R2, (precio e1 9) (peso e1 2.2)})

El principio de refracción impediría la ejecución repetida de la misma instancia de la regla R1, pero no de la regla R2, ya que la instancia que existe ahora en la BH es nueva y diferente de la instancia en el ciclo previo 2; por lo que la instancia de la regla 2 se seleccionaría y ejecutaría, incrementando nuevamente el coste del envío en 2 euros más. Este proceso de repetiría indefinidamente. Como conclusión, podemos afirmar que la regla R2 sigue incorrectamente diseñada. Para evitar que se dispare repetidas veces dicha regla, una solución consiste en eliminar de la BH el peso del envío, en el momento que se haya ejecutado por primera vez la regla R2; ya que a partir de ese punto de la inferencia, ya no se necesitará conocer el peso del envío. Esto significa añadir una nueva cláusula a la parte de acción de la regla R2:

SI (precio ?e ?p) \wedge (peso ?e ?k) \wedge (test ($>?k$ 2))
ENTONCES

Añadir(precio ?e ?p+ (?k-2)*10) Borrar (precio ?e ?p)
(Borrar (peso ?e ?k))

- (c) La regla R3 debe incrementar el precio del envío con el suplemento adecuado, si el envío es urgente. Por la analogía con la regla R2, la regla R3 sería:

SI (precio ?e ?p) \wedge (entrega ?e urgente)
ENTONCES
Añadir(precio ?e ?p+6) Borrar (precio ?e ?p)
(Borrar (entrega ?e urgente))

■

3.10 Ejercicios propuestos

3.1. Obténgase la red de inferencia para la siguiente Base de Reglas de un sistema de clasificación de animales:

R1: Si un animal tiene pelo, entonces es mamífero.

R2: Si un animal da leche, entonces es mamífero.

R3: Si un animal tiene plumas es un ave.

R4: Si un animal vuela y pone huevos, es ave.

R5: Si un animal come carne, es carnívoro.

R6: Si un animal tiene dientes puntiagudos, tiene garras, tiene ojos al frente es carnívoro.

R7: Si un animal mamífero tiene pezuñas es una ungulado.

R8: Si un animal mamífero rumia es un ungulado.

R9: Si un animal mamífero y carnívoro tiene color leonado con manchas oscuras se trata de un leopardo.

R10: Si un animal mamífero y carnívoro tiene color leonado con rayas negras es un tigre.

R11: Si un animal ungulado con cuello largo y piernas largas tienen manchas oscuras es una jirafa.

R12: Si un animal es un ungulado con rayas negras es una cebra.

R13: Si un animal es ave y no vuela y tiene el cuello largo y piernas largas de color blanco y negro es un avestruz.

R14: Si un animal es ave, no vuela, nada, de color blanco y negro, se trata de un pingüino.

R15: Si es un ave que vuela bien, es un albatros.

R16: Si un animal es de una especie y ese animal es padre de otro, entonces el hijo es de la misma especie.

Teniendo en cuenta la siguiente BH:

(animal robbie) (robbie manchas oscuras) (robbie come carne) (suzie tiene plumas)
(suzie vuela bien)

¿Qué nuevos hechos se pueden deducir aplicando un encadenamiento hacia delante a la red de inferencia obtenida previamente?

3.2. Obténgase la red de inferencia para la siguiente Base de Reglas simbólicas:

$$\begin{aligned} R1: \quad & J \rightarrow A \\ R2: \quad & D \rightarrow A \\ R3: \quad & D \wedge F \rightarrow B \\ R4: \quad & H \wedge I \wedge J \rightarrow B \\ R5: \quad & K \wedge H \rightarrow D \\ R6: \quad & C \wedge F \rightarrow H \\ R7: \quad & G \wedge F \wedge I \rightarrow D \\ R8: \quad & A \wedge B \wedge C \rightarrow M \\ R9: \quad & C \rightarrow I \end{aligned}$$

Téngase en cuenta la siguiente BH:

$$BH = \{C \ F \ G\}$$

$$\text{Meta: } M$$

Supóngase que la regla 9 es la más prioritaria y la 1 la menos prioritaria.

1. Aplique un encadenamiento hacia delante, siguiendo como estrategia de resolución del conjunto conflicto el criterio de prioridad establecido.
2. Aplique un encadenamiento hacia delante, siguiendo como estrategia de resolución del conjunto conflicto primero el criterio de prioridad y segundo el principio de refracción.
3. Aplique un encadenamiento hacia atrás siguiendo como estrategia de resolución del conjunto conflicto el criterio de prioridad.

3.3. Una tienda de coches tiene un portal que aconseja a sus clientes qué coche comprar en función de sus preferencias. La información sobre los modelos de coches que se pueden comprar se muestra en la Tabla 3.4. El portal proporciona a los clientes un formulario con las siguientes preguntas:

1. ¿Qué cantidad en euros desea gastarse?
2. ¿Busca un maletero pequeño, mediano o grande?
3. Como mínimo, ¿cuántos caballos debe tener el coche?
4. ¿Desea ABS?

5. Como máximo, ¿qué consumo debe tener el coche a los 100 km?

El usuario puede responder a todas las preguntas del formulario, pero también puede dejar algunas preguntas sin responder. En dicho caso, los valores por defecto que se considerarán son los siguientes:

1. El precio del coche no debe superar los 13.000 euros.
2. El maletero del coche debe ser grande.
3. El coche debe tener 80 caballos, como mínimo.
4. El coche debe estar dotado de ABS.
5. El consumo a los 100 km debe ser, como máximo, 8 litros.

Una vez rellenado y enviado el formulario a la tienda virtual, el sistema introduce las preferencias del cliente en su BH e infiere qué coche comprar. Diseñe un SBR para gestionar dicha tienda virtual. Para ello, se pide:

1. Represéntese la BH inicial y descríbese sus contenidos iniciales.
2. Diséñese el conjunto de reglas necesarias para gestionar la tienda virtual.
3. Indíquese la estrategia de control más apropiada.
4. Supóngase que un cliente desea un coche con un maletero grande que no consuma más de 9 litros a los 100 Km, y no contesta al resto de preguntas. Indíquese cómo se representa esta información en la BH. Explique qué conclusión alcanzaría el SBR si se utilizan los valores por defecto y no se considera la *hipótesis del mundo cerrado* (véase 3.4.1).
5. Supóngase el mismo caso que el anterior y explique qué conclusión alcanzaría el SBR si no se utilizan los valores por defecto y se supone la *hipótesis del mundo cerrado*.

MODELO	PRECIO en euros	TAMANO de MALETERO	NÚMERO de CABALLOS	ABS	CONSUMO en litros
Modelo-1	12.000	pequeño	65	no	4,7
Modelo-2	12.500	pequeño	80	sí	4,9
Modelo-3	13.000	mediano	100	sí	7,8
Modelo-4	14.000	grande	125	sí	6,0
Modelo-5	15.000	pequeño	147	sí	8,5

Tabla 3.4: Características de los coches.

Referencias

- BORRAJO, D.; MARTÍNEZ, V.; JURISTO, N. y PAZOS, J.: *Inteligencia Artificial. Métodos y Técnicas*. Editorial Centro de Estudios Ramón Areces, Madrid, 1993.
- CALVO, A.; GONZÁLEZ, P.; ROMERO, C. y VENTURA, S.: *Programación en lenguaje Clips*. Editorial Universitaria Ramón Areces, Madrid, 2005.
- FERNÁNDEZ, S.; GONZÁLEZ, J. y MIRA, J.: *Problemas resueltos de Inteligencia Artificial*. Addison Wesley, 1998.
- STEELE, G.L.: *Common Lisp the Language*. <http://www-2.cs.cmu.edu/~Groups/AI/html/cltl/cltl2.html>, 2^a edición, 1990.
- WINSTON, P.R.: *Inteligencia Artificial*. Addison-Wesley Iberoamericana, Massachusetts, 3^a edición, 1994.

Capítulo 4

Redes semánticas y marcos

María del Carmen Suárez de Figueroa Baonza y Asunción Gómez Pérez
Universidad Politécnica de Madrid

4.1 Introducción

En representación de conocimientos se puede hablar de tres modelos distintos:

- El *modelo conceptual* que es una representación de los conocimientos de un dominio, independientemente de cómo se implementen, utilizando estructuras no computables que modelizan el problema y la solución en un dominio concreto.
- El *modelo formal* que es una representación “semiinterna” o “semicomputable” de los conocimientos de un dominio. Este modelo formal se ha de obtener a partir del modelo conceptual.
- El *modelo computable* que hace que el modelo formal sea totalmente operativo, y que está formado por una base de conocimientos, un motor de inferencias y una serie de estrategias de control.

El presente capítulo está relacionado con la formalización de conocimientos. A grandes rasgos, se puede decir que **formalizar** consiste en *representar* simbólicamente los conocimientos de un dominio utilizando alguno de los formalismos de representación de conocimientos existentes, *organizarlos* de acuerdo a algún modelo de diseño y *determinar los métodos de inferencia adecuados* para manejar eficiente y efectivamente los conocimientos representados.

Para representar los conocimientos de un determinado dominio, el ingeniero del conocimiento (IC) debe utilizar uno o varios formalismos o técnicas de representación de conocimiento. Para razonar con los conocimientos representados, el IC debe analizar las técnicas de inferencia disponibles para cada formalismo utilizado y seleccionar las más adecuadas para resolver el problema que se plantea.

Los investigadores en IA han descrito varios formalismos útiles para la representación de conocimientos, cada uno de ellos más adaptado a un tipo de conocimientos. Los formalismos más populares incluyen colecciones de reglas condicionales, del tipo

“*Si ..., Entonces ...*” (véase el capítulo 3). Otros sistemas de representación recurren a *marcos conceptuales*, estructuras parecidas a formularios que es preciso llenar. Existen otros métodos de representación que se valen de redes “en telaraña” o de listas, como las *redes semánticas* y los *guiones*, respectivamente.

Cuando un IC aborda un nuevo problema, una de las decisiones que tiene que tomar es determinar el formalismo de representación más adecuado para la tarea que le corresponde realizar al sistema. En sistemas pequeños y sencillos, lo normal es que solamente se utilice un formalismo para representar el conocimiento; pero a menudo se combinan dos o tres formalismos de modo que se complementen unos a otros para lograr una representación que se ajuste de la forma más natural posible a los conocimientos del dominio.

Cada uno de los formalismos existentes tiene asociadas unas técnicas básicas de inferencia. Estas técnicas son independientes del dominio representado por el formalismo, es decir, serán capaces de razonar con cualquier conjunto de conocimientos representados mediante su formalismo propietario. Básicamente, los distintos formalismos existentes permiten representar cualquier conocimiento; sin embargo, unos formalismos son más adecuados que otros para representar unos tipos de conocimientos. En función de si el formalismo es más adecuado para representar conceptos, relaciones o acciones, se puede realizar la siguiente clasificación:

- **Formalismos basados en conceptos**, los cuales representan las principales clases o entidades del dominio, sus propiedades, y los posibles valores que puede tomar cada propiedad. En este caso, se pueden distinguir entre las ternas *Objeto-Atributo-Valor* y los *Marcos*
- **Formalismos basados en relaciones**, que centran su atención en las relaciones que aparecen entre los conceptos o entidades del dominio. Los más importantes son: la *Lógica* (véase el capítulo 2), las *Redes Semánticas* y la *Teoría de la Dependencia Conceptual*.
- **Formalismos basados en acciones**, que describen los conocimientos del dominio como un conjunto de acciones básicas. Los principales formalismos de este tipo son: los *Sistemas de Producción* (véase el capítulo 3) y los *Guiones*.

Este capítulo se centra en un formalismo basado en relaciones, las *redes semánticas*, y en un formalismo basado en conceptos, los *marcos*.

El formalismo de las **Redes Semánticas** fue definido por Quillian [Quillian, 1968] como un grafo orientado formado por nodos y arcos unidireccionales, ambos etiquetados. Los nodos representan conceptos y los arcos relaciones entre conceptos. Las redes semánticas se están utilizando en IA como una técnica de representación de conocimientos para expresar relaciones entre los conceptos de un dominio. El término *red semántica* se ha utilizado para describir una gran variedad de representaciones y técnicas que tienen en común el uso de grafos orientados como representación gráfica. Dado que no existe un formalismo universalmente aceptado que permita explicar con claridad qué es una red semántica, en este capítulo se van a mostrar los trabajos más representativos realizados en este área tanto en el plano de representación como en el plano del razonamiento.

El formalismo de **Marcos** (en inglés *Frames*), fue definido por Minsky [Minsky, 1985] como una estructura de datos para representar estereotipos. Cada marco está formado por un nombre y un conjunto de propiedades. Los valores de cada propiedad se describen en el marco utilizando un conjunto de facetas. Los marcos se unen unos con otros mediante relaciones. La relación *subclase-de* permite crear jerarquías de conceptos por especialización y proporciona un camino para la herencia de propiedades. Las principales características que han hecho de los marcos el formalismo de representación más utilizado cuando los conocimientos del domino están basados en conceptos son: la organización de los conceptos que intervienen en el problema en jerarquías o taxonomías de conceptos, y la posibilidad de representar conocimientos procedimentales y declarativos en los marcos de una forma integrada.

4.2 Redes semánticas

Las redes semánticas (o redes conceptuales) son un formalismo o paradigma de representación de conocimiento basado en relaciones; es decir, este formalismo centra su atención en las relaciones que aparecen entre los conceptos o entidades de un dominio. Básicamente, una red semántica [Quillian, 1968] es un grafo orientado formado por nodos etiquetados, que representan conceptos e instancias, y arcos unidireccionales etiquetados, que representan relaciones entre conceptos o instancias. La Figura 4.1 muestra los conceptos básicos de representación en redes semánticas.

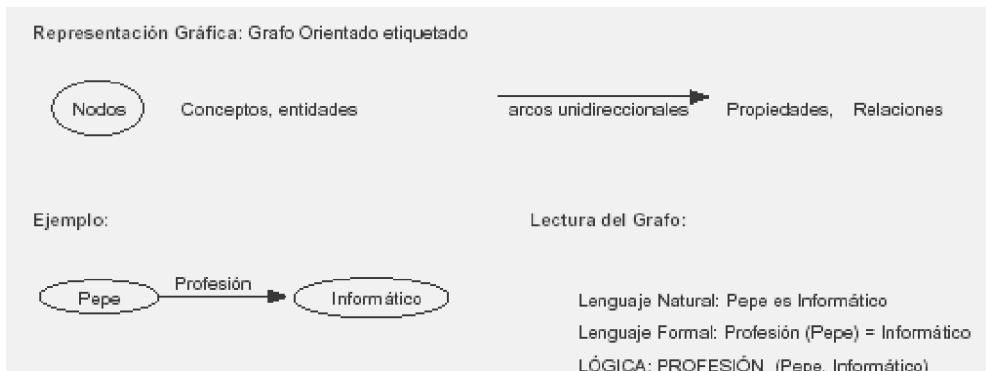


Figura 4.1: Conceptos básicos en redes semánticas.

4.2.1 Representación de conocimiento

En una red semántica, la información se representa en un grafo orientado que está formado por un conjunto de **nodos** y **arcos** unidireccionales, ambos etiquetados. Los nodos representan conceptos e instancias de dichos conceptos, y los arcos, que conectan nodos, representan relaciones binarias entre ellos. Por tanto, el significado de un concepto de la red dependerá de la forma en la que dicho concepto se relaciona con

otros conceptos. Se puede utilizar cualquier etiqueta para dar nombre a cualquier nodo o arco. Los principales problemas de este formalismo de representación son la falta de una terminología adecuada y universalmente aceptada, y de una semántica uniforme y precisa. Sin embargo, su gran atractivo gráfico y su intuitiva interpretación las hace rápidamente comprensibles y utilizadas en la formalización de bases de conocimiento.

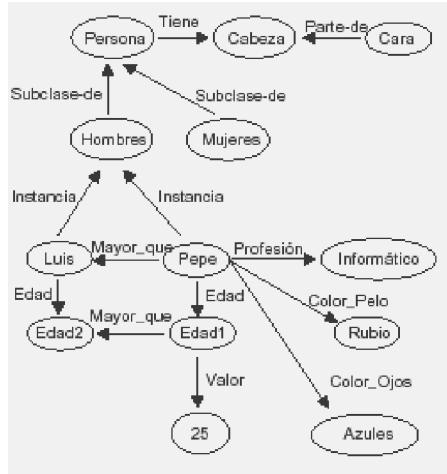


Figura 4.2: Ejemplo de red semántica.

La Figura 4.2 es un ejemplo sencillo de una red conceptual que representa a *Pepe*, un profesional de la informática que tiene los ojos azules, el pelo rubio, 25 años y que es mayor que Luis. La base de la representación de conocimientos en las redes semánticas consiste en modelar los conocimientos relativos a un objeto o concepto mediante pares atributo-valor. Los pares se representan en el grafo orientado de la siguiente manera: el nodo origen (*Pepe*) es el objeto o concepto para el cual se definen los pares atributo-valor, los arcos que parten de dicho nodo (*profesión*, *color-pelo*, etc.) son los atributos del par, y los nodos destinos (*informático*, *rubio*, etc.) representan los valores de los atributos. Por tanto, el significado de un nodo en la red de conceptos dependerá no sólo de cómo el nodo se relaciona con otros nodos, sino de las etiquetas que dan nombre a los arcos y nodos que representan los elementos del dominio. El ingeniero del conocimiento podrá utilizar tantos arcos como propiedades estime oportuno representar y, a cada uno de ellos, le asignará una etiqueta que expresará de manera representativa su semántica.

Básicamente, los arcos en las redes conceptuales se agrupan en dos categorías: **arcos descriptivos** y **arcos estructurales**.

- Los **arcos descriptivos** describen entidades y conceptos. Ejemplos de arcos descriptivos en la red de la Figura 4.2 son *profesión* y *color-pelo*. En la red conceptual de la misma figura podemos ver dos tipos de arcos descriptivos: arcos que relacionen dos entidades independientes ya existentes y arcos utilizados para

definir una nueva entidad. Por ejemplo, el arco *profesión* une dos nodos (*Pepe, informático*) con existencia propia. Sin embargo, los nodos *Edad1* y *Edad2* son nuevos conceptos que representan la edad de *Pepe* y de *Luis* respectivamente, y que se definen por sus relaciones con dichos nodos.

- Los **arcos estructurales** enlazan las entidades o conceptos formando la arquitectura o estructura de la red. A diferencia de los arcos descriptivos, la semántica de los arcos estructurales es independiente de los conocimientos del dominio que se está representando. Ejemplos de arcos estructurales en la Figura 4.2 son los etiquetados como *subclase-de*, *instancia* y *parte-de*, que se corresponden respectivamente con los procesos básicos de **generalización**, **instanciación**, y **agregación**. No obstante, el IC puede definir, a medida, tantas etiquetas estructurales como crea oportuno. Estos procesos básicos asociados a arcos estructurales se pueden definir de la siguiente forma:
 - La **generalización** pone en relación una clase con otra más general, formando una red de nodos por especialización de conceptos. Las propiedades definidas en los nodos generales se heredan por deducción en los nodos específicos, siguiendo los arcos *subclase-de*. Por ejemplo en la Figura 4.2 se puede deducir que *Pepe* por ser *Hombre* y por ser *Persona* tiene *Cabeza*.
 - El arco **instanciación** liga un objeto concreto con su tipo genérico. Por ejemplo, la aserción “*Pepe es un Hombre*” se representa en la Figura 4.2 mediante el arco *instancia* desde el nodo *Pepe* hacia el nodo *Hombre*.
 - La **agregación** liga un objeto con sus componentes. Siguiendo con el ejemplo de la figura 4.2, la aserción “*la Cara forma parte de la Cabeza*” se representa utilizando el arco *parte-de* desde el nodo *Cara* hacia el nodo *Cabeza*.

Alternativamente a esta representación gráfica, los conocimientos expresados en una red semántica también pueden expresarse utilizando lógica proposicional y cálculo de predicados de primer orden (véase el capítulo 2). Por ejemplo, teniendo en cuenta el conocimiento representado en la Figura 4.2 se pueden definir los siguientes predicados:

$$\begin{aligned}
 & \text{PROFESIÓN}(\textit{Pepe, informático}) \\
 & \text{INSTANCIA}(\textit{Pepe, Hombre}) \\
 & \text{SUBCLASE - DE}(\textit{Hombre, Persona}) \text{ o } \forall x \text{ Hombre}(x) \Rightarrow \text{Persona}(x) \\
 & \text{TIENE}(\textit{PersonaCabeza})
 \end{aligned}$$

Sin embargo, predicados *n*-arios (es decir de aridad superior a dos), como por ejemplo *COMPRA-VENTA*(*Pepe, Luis, reloj1, 45, euros*), que representa la información de que “*Pepe compra a Luis un reloj por 45 euros*”, no pueden ser representados en estas redes conceptuales. Además, si en la red de la Figura 4.2 el IC quisiera representar el predicado *VIO*(*Pepe, museo*), que representa que “*Pepe vio un museo*”, se estarían mezclando en la red arcos que representan atributos (*profesión, color-pelo*,

color-ojos y edad) con el arco que representa la acción *vio*. Esto, que inicialmente parece no crear ningún problema en la representación de conocimiento, desde el punto de vista de la inferencia sí que ocasiona problemas, ya que el razonador deduciría que *vio* es un atributo de *Pepe* mientras que *vio* es una acción realizada por *Pepe* y, por ello, tendría que tener un tratamiento diferente.

4.2.2 Representación de predicados no binarios

Los predicados de aridad distinta a dos se pueden representar también en redes semánticas. Por ejemplo, el predicado de aridad uno *HOMBRE(Pepe)* equivaldría a *INSTANCIA(Pepe, Hombre)*. El problema aparece cuando se utilizan predicados de aridad tres o superior. En este caso, un nuevo objeto representa al predicado de aridad mayor que dos, y nuevos predicados binarios describen las relaciones entre este nuevo objeto y sus argumentos. Por ejemplo, al representar el predicado *COMPRA-VENTA(Pepe, Luis, Reloj1, 45, euros)* en una red semántica, se crearía un nodo que representa la compra-venta (*compra-venta1*) y cinco predicados (*vendedor*, *comprador*, *objeto*, *precio*, *unidad*) que representan las relaciones con las cinco piezas de información, como se muestra en la Figura 4.3.

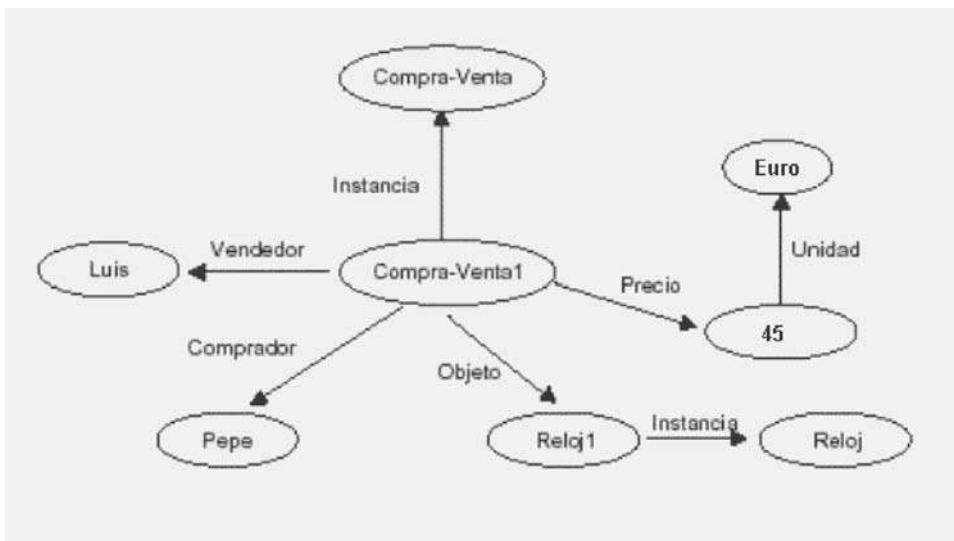


Figura 4.3: Representación de predicados no binarios.

4.2.3 Representación de acciones

La adaptación de conceptos de la gramática de casos de Fillmore [Fillmore, 1968] a las redes conceptuales permite al IC representar acciones. La gramática de casos se basa en que toda proposición contenida en una sentencia tiene una estructura

profunda formada por un verbo, que es el elemento principal, y una o más frases nominales. Cada frase nominal se relaciona con el verbo mediante un conjunto de casos. Los **casos** hacen referencia al:

- *Agente*: persona que realiza el evento.
- *Contra-agente*: fuerza o resistencia contra la que se ejecuta la acción.
- *Objeto*: la entidad que es movida, cambiada, o cuya posición o existencia se considera.
- *Resultado*: la entidad que aparece como consecuencia de la acción.
- *Instrumento*: el estímulo o causa física inmediata de un evento.
- *Origen*: el lugar del que procede el evento.
- *Propósito*: el motivo por el que se ejecuta la acción.
- *Lugar*: sitio en el que se desarrolla la acción.
- *Tiempo*: fecha o momento en el que tiene lugar la acción.
- *Sujeto*: entidad que recibe, acepta, experimenta o sufre el efecto de la acción.

La **modalidad**, por su parte, hace referencia a características que presenta el verbo, tales como:

- El *tiempo* en el que se ha desarrollado la acción (presente, pasado, futuro).
- La *voz* del verbo: activa o pasiva.

Un análisis detallado de cada verbo determina los casos que normalmente tiene asociados, permitiendo construir un patrón con sus casos obligatorios y con sus casosopcionales. El patrón se instanciará o se llenará con los valores que toma cada caso o situación concreta.

Utilizando la información proporcionada por la gramática de casos, para representar afirmaciones que no se refieren a atributos de los objetos, sino a acciones y eventos, se puede representar la afirmación “*Pepe vio el Prado en Madrid*” en una red semántica utilizando nodos *situación* o *suceso*, tal y como se muestra en la Figura 4.4. Cada nodo situación tiene como atributos el conjunto de casos (*agente*, *objeto*, *lugar*) y de modalidades (*tiempo*, *voz*) que describen el evento, siendo el valor de cada atributo el valor de cada caso, (*Pepe*, *El Prado*, *Madrid*), o de cada modalidad (*Pasado*, *Activa*). Con esta representación es relativamente sencillo representar acciones o sentencias. Además, las sentencias se pueden representar en lógica utilizando un predicado de aridad mayor que dos como *VIO(Pepe, El Prado, Madrid)*. Finalmente, mencionar que el esquema de nodos situación permite representar sentencias compuestas como “*Luis sabe que Pepe vio El Prado en Madrid*”, tal y como se muestra en la Figura 4.4.

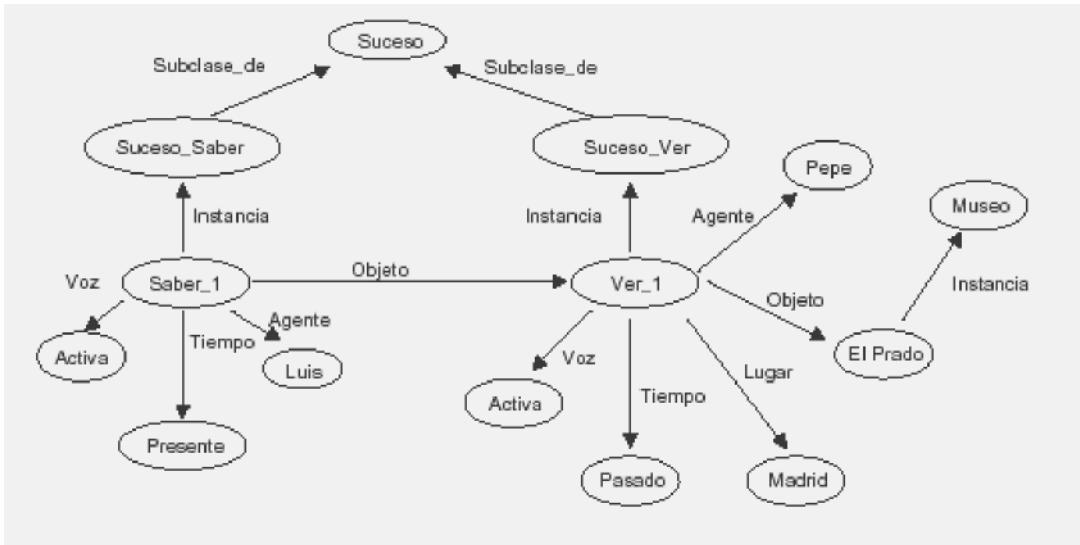


Figura 4.4: Representación de acciones en redes semánticas.

4.2.4 Representación de conocimiento disjunto

Hasta este momento se han analizado las redes semánticas para representar objetos y conceptos no excluyentes. Sin embargo, en ciertas ocasiones, el IC conoce qué entidades del dominio son disjuntas entre sí, y este conocimiento también debe expresarse en la red semántica. Conceptos o situaciones disjuntas pueden representarse en la red semántica utilizando la etiqueta disjunto (véase la Figura 4.5).

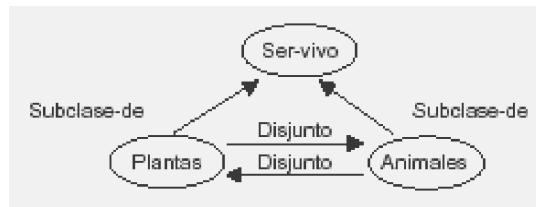


Figura 4.5: Representación de conocimiento disjunto en redes semánticas.

Sin embargo, Hendrix [Hendrix, 1975, 1979] dotó al formalismo de unos arcos especiales que permiten representar estos conocimientos de manera más sencilla, como se muestra en la Figura 4.6. Las etiquetas utilizadas en estos arcos especiales son las siguientes:

- S : Subconjunto.
- SD : Subconjunto disjunto.
- E : Elemento.
- ED : Elemento disjunto.

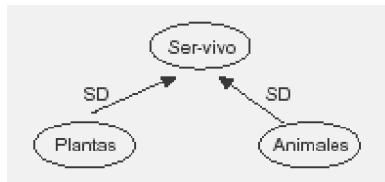


Figura 4.6: Representación de conocimiento disjunto en redes semánticas utilizando etiquetas.

4.3 Inferencia de conocimiento en redes semánticas

Un sistema que utilice como formalismo de representación de conocimientos las redes semánticas, debe utilizar los conocimientos almacenados en la red para resolver los casos que se planteen. La eficacia del razonamiento en las redes depende de los procedimientos que trabajan con la semántica de sus arcos. Las técnicas más empleadas son la **equiparación** y la **herencia de propiedades**.

4.3.1 Equiparación

Se dice que un fragmento de red, apunte o consulta se equipara con una red semántica, si el apunte se puede asociar con un fragmento de la red semántica. Para explicar la técnica de equiparación en redes semánticas se utilizará como base de conocimientos la red de la Figura 4.4, y la consulta “*¿Existe algún varón que viera un museo en Madrid?*”. Los pasos a seguir en el proceso de equiparación son:

1. Se construye un apunte que responda a la pregunta en cuestión utilizando los criterios seguidos en la construcción de la red semántica. El apunte está formado por un conjunto de **nodos constantes**, **nodos variables** y **arcos etiquetados**. Los nodos constantes son los datos conocidos de la pregunta; en nuestro ejemplo, *varón*, *Madrid*, *museo*, *suceso-ver*; los nodos variables son los valores que se requieren y, por consiguiente, son desconocidos, *Ver-?* y *Varón?*; y los arcos *instancia*, *agente*, *lugar*, *objeto* unen nodos constantes y nodos variables entre sí. El apunte descrito se muestra en la Figura 4.7.
2. Se coteja o superpone el apunte sobre la red semántica en la base de conocimientos.

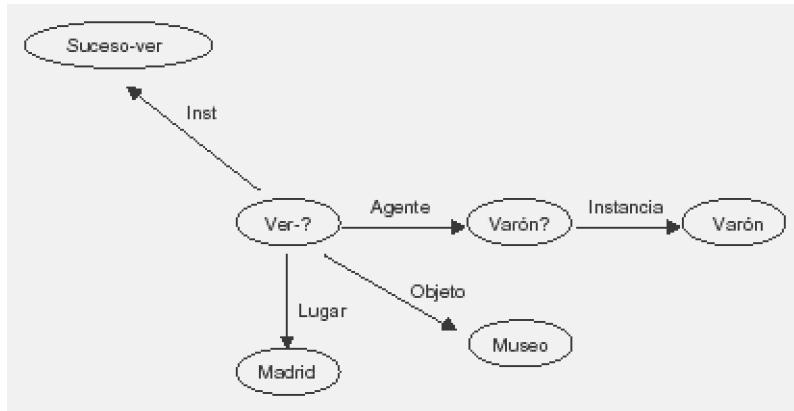


Figura 4.7: Apunte para la consulta *¿hay algún varón que viera un museo en Madrid?*

3. Los nodos variables se ligan a los nodos constantes de la red hasta encontrar una equiparación perfecta. Con este fin, en el ejemplo de la Figura 4.4, se busca en la red un nodo situación del que parte un arco *Lugar* hacia un nodo *Madrid*, y un arco *Instancia* hacia el nodo *Suceso-Ver*. Cuando esto ocurre, se liga la variable *Ver-?* del apunte con el nodo situación de la red, *Ver-1*, y el nodo *Varón?* del apunte con el nodo constante de la red *Pepe*. Dado que en el apunte se conoce que se trata de un varón, es necesario comprobar en la red que desde *Pepe* parte un arco *Instancia* hacia el nodo *Varón* (u *Hombre*).
4. La respuesta a la consulta es el fragmento de red semántica con los valores con los que se rellenan los nodos variables. En el ejemplo: *Ver-? = Ver-1* y *Varón? = Pepe*. Si no hubiera ninguna equiparación del apunte con la red semántica, la respuesta dada por la técnica de equiparación sería “*No se ha encontrado un varón que haya visto un museo en Madrid*”.

4.3.2 Herencia de propiedades

El concepto de herencia de propiedades tiene su fundamento teórico en la regla del **modus ponens**. La herencia de propiedades permite que nodos específicos de una red accedan a las propiedades definidas en otros nodos utilizando los arcos *Instancia* y *Subclase-de*, favoreciendo así la compartición de propiedades entre diferentes nodos y evitando la repetición de propiedades en la base de conocimiento. La herencia de propiedades se puede utilizar en sistemas que razonan dirigidos por la meta o por los datos.

Supóngase que se quiere determinar la veracidad de la sentencia “*Dumbo es de color gris*”, en la red semántica de la Figura 4.8. En primer lugar se debe localizar el nodo *Dumbo*, y después se debe buscar si desde dicho nodo sale un arco con la etiqueta *Color*. Al no existir el susodicho arco en *Dumbo*, el sistema buscará arcos *Instancia* que partan desde dicho nodo hacia algún otro nodo. En este caso, existen

dos arcos con destino *Elefante* y *Macho*, y el motor de inferencias elegirá alguna de las dos alternativas. Si elige el nodo *Elefante*, como desde *Elefante* parte el arco buscado hacia el nodo *Gris*, entonces el motor de inferencias devolverá que la sentencia “*Dumbo es de color gris*” es cierta. Pero si se elige el nodo *Macho*, al no encontrarse la propiedad en él, utilizando el arco *Subclase-de* se buscará si dicha etiqueta se encuentra en sus *superclases*. Dado que en *Animal* no se encuentra la propiedad y que en *Ser_Vivo* tampoco, entonces se retrodecerá y se empezará a explorar el resto de alternativas. Si una vez que se han explorado todas las alternativas, no se encuentra la solución, entonces se debe comunicar al usuario que, con la información almacenada en la red semántica, no se puede emitir un juicio sobre la verdad o falsedad de la sentencia. Sabiendo que Dumbo es un Elefante, se podría deducir que “*Dumbo es gris*”, que “*Dumbo necesita oxígeno*” y que “*Dumbo es un ser vivo*”. Por consideraciones de eficiencia, estos hechos no se almacenan explícitamente en la red, puesto que se pueden deducir del resto de conocimiento almacenado.

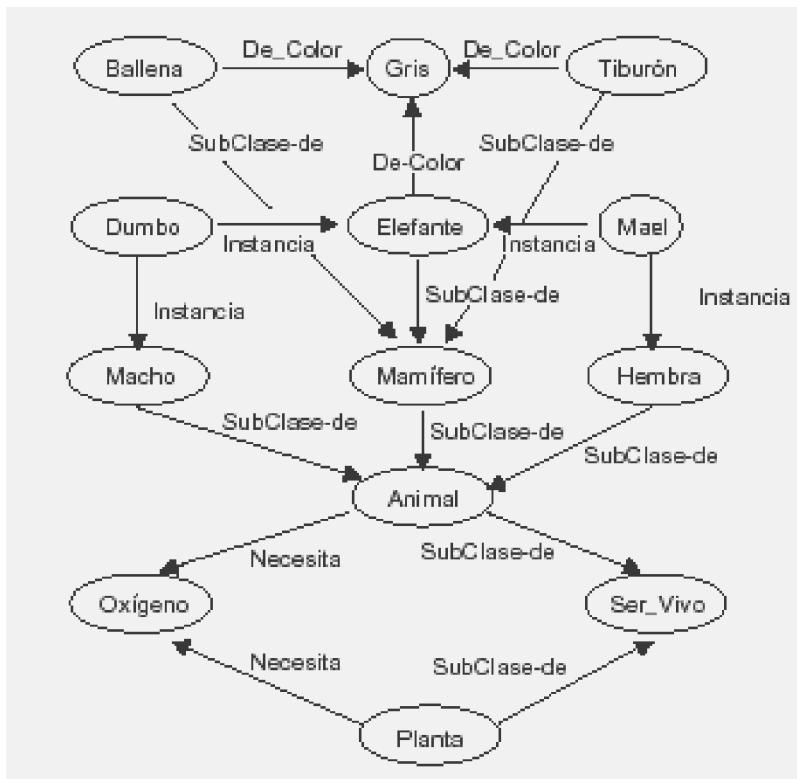


Figura 4.8: Red semántica que describe a Dumbo.

La herencia de propiedades trabaja muy bien con propiedades que presentan excepciones en sus valores. La distribución de las propiedades en la red permite que se herede el valor de la propiedad del nodo más cercano al nodo que sirvió como punto

de partida en la inferencia. Por ejemplo, si en la red de la Figura 4.9 se pregunta por el color de *Brutus*, utilizando la herencia de propiedades se deduce que “*Brutus es de color Negro*”. Sin embargo, si se pregunta por el color de *Copito de Nieve*, el valor que se obtiene es *Blanco*, porque desde *Copito de Nieve* sale un arco etiquetado con color hacia el nodo *Blanco*.

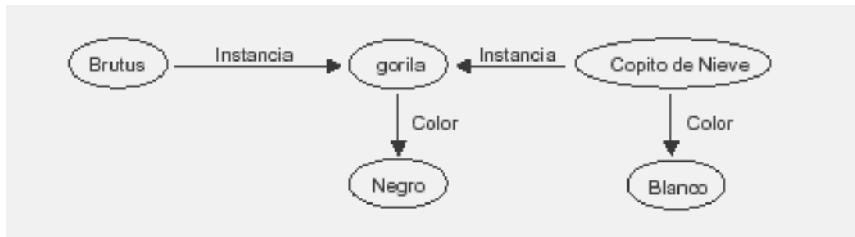


Figura 4.9: Ejemplo de tratamiento de excepciones.

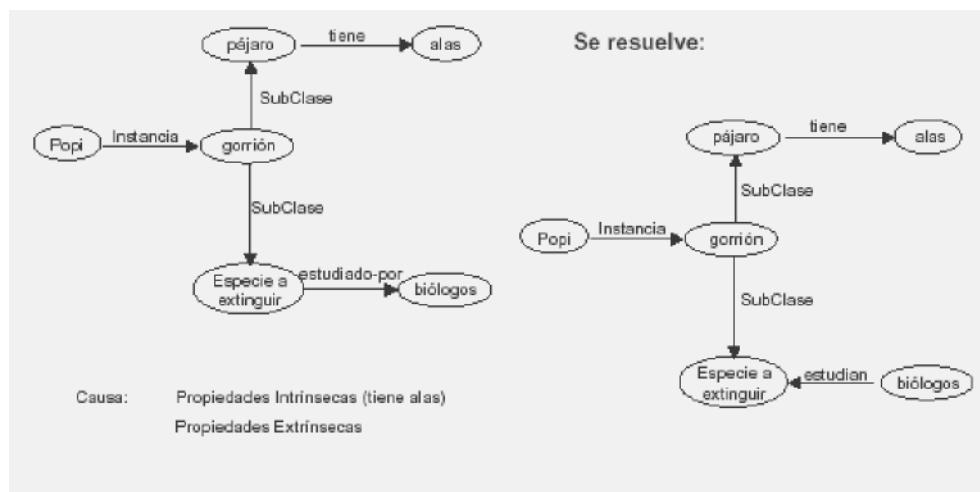


Figura 4.10: Problema en la aplicación de herencia de propiedades.

La inferencia basada en herencia de propiedades origina problemas si el IC ha formalizado mal los conocimientos. Los principales errores que se suelen cometer son:

- No distinguir los nodos que son instancias de aquellos que son conceptos.
- La etiqueta que da nombre al nodo o al arco tiene una semántica diferente al conocimiento que se quiere representar.
- El arco está en sentido contrario.
- No se han representado situaciones o acciones utilizando nodos situación.

En la red de la Figura 4.10, el problema aparece al aplicar herencia de propiedades. En este ejemplo, *Popi* hereda todas las propiedades de *Especie a Extinguir*, del mismo modo que hereda las de *Pájaro*, lo que puede ser cierto o no. La diferencia fundamental se encuentra en la naturaleza de los arcos *tiene* y *estudiado-por*, puesto que mientras que el primero se refiere a características intrínsecas al concepto *Pájaro*, el segundo no constituye un requisito para ser *Pájaro*. Por consiguiente, el problema se encuentra en el sentido del arco y en la etiqueta *estudiado-por*, y no en la técnica de inferencia. Al cambiar el sentido del arco y la etiqueta *estudiado-por* por la etiqueta *estudian*, el problema se resuelve.

4.4 Marcos

El formalismo de marcos se ha revelado como la técnica de representación de conocimientos más utilizada en IA cuando los conocimientos del dominio están organizados sobre la base de conceptos. Minsky [Minsky, 1985] definió los **marcos** como una estructura de datos que representa situaciones estereotipadas construidas sobre situaciones similares ocurridas anteriormente, permitiendo así aplicar a situaciones nuevas los conocimientos de situaciones, eventos y conceptos previos. Los conocimientos que se expresan en los marcos son conocimientos declarativos del dominio. Dentro del marco, existen conocimientos procedimentales que se refieren a: cómo utilizar el marco, qué se espera que suceda a continuación, así como el conjunto de acciones que se deben realizar tanto si las expectativas se cumplen como si éstas fallan. Los marcos organizan los conocimientos del dominio en árboles, también llamados jerarquías, o en grafos, ambos construidos por especialización de conceptos generales en conceptos más específicos.

Las técnicas de inferencia utilizadas para razonar con los conocimientos de la base de conocimientos son: *equiparación*, para clasificar entidades en una jerarquía; *herencia simple* y *herencia múltiple*, para compartir propiedades que están distribuidas en la jerarquía de conceptos o en el grafo, respectivamente; y *valores activos* y *métodos* para representar la conducta del sistema.

4.4.1 Representación de conocimiento

Los conceptos que el IC utilizará al formalizar la base de conocimientos en marcos son: **marcos** para representar conceptos o elementos, **relaciones** para expresar dependencias entre conceptos, **propiedades** para describir cada concepto, y **facetas** para expresar de múltiples formas los valores con los que se puede llenar cada propiedad. La Figura 4.11 muestra los conceptos básicos de representación en marcos.

4.4.1.1 Representación de conceptos e instancias

Básicamente, existen dos tipos de marcos: los **marcos clase** y los **marcos instancia**:

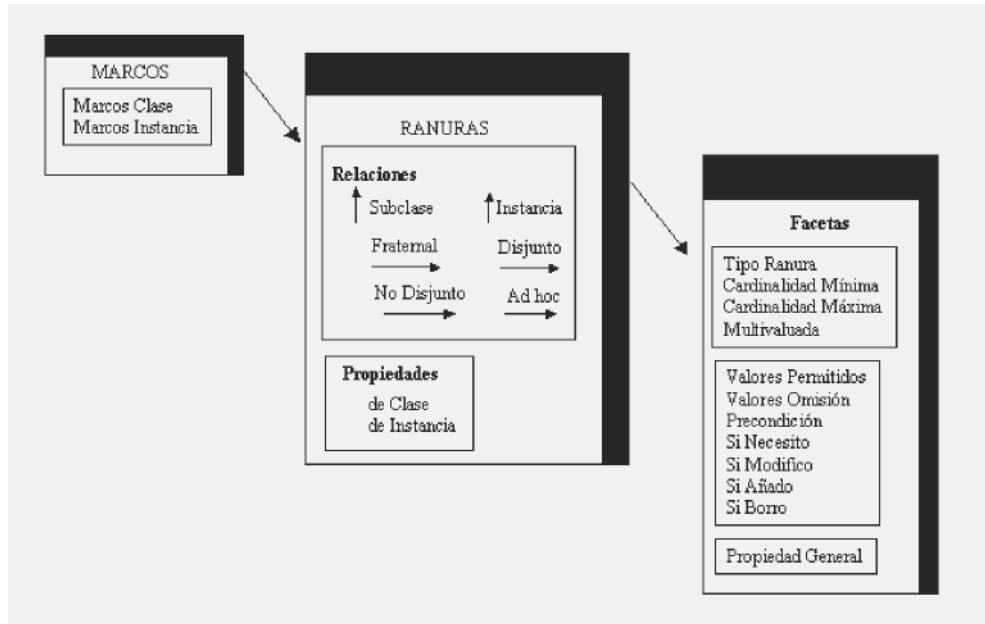


Figura 4.11: Conceptos básicos en marcos.

- Los *marcos clase* se utilizan para representar conceptos, clases o situaciones genéricas descritos por un conjunto de propiedades, unas con valores y otras sin valores asignados, que son comunes al concepto, clase o situación que el marco representa. Los marcos clase representan conceptos, es decir, entidades acerca de las cuales se desea describir cierto tipo de información. Los conceptos pueden ser de cualquier índole, ya se refieran a entidades físicas tangibles, descripción de tareas, procesos de razonamiento, entidades abstractas, etc. Los marcos *Persona*, *Hombre* y *Mujer* de la Figura 4.12 son ejemplos de marcos clase.
- En los dominios de trabajo de los expertos, existen elementos, instancias, o individuos de clases. Por ejemplo, el *martillo-1* de la clase herramienta para el experto carpintero. El formalismo de marcos permite representar estos objetos utilizando los *marcos instancia*. Los marcos instancia pueden considerarse como la representación en el dominio real de una clase determinada. Estos marcos instancia deben estar relacionados, como mínimo, con un marco clase. Además, han llenado la mayoría de sus propiedades con valores específicos; es decir, con información concreta del elemento, instancia o individuo que representan, y que pertenece al concepto, clase o situación representado por un marco clase. El resto de propiedades que no estén almacenadas en él, las hereda de los marcos clase de los cuales son instancias. Los marcos *María*, *Ana*, *Luis*, *Pepe* y *Juan* son marcos instancia de los marcos clase *Mujer* y *Hombre* en la Figura 4.12.

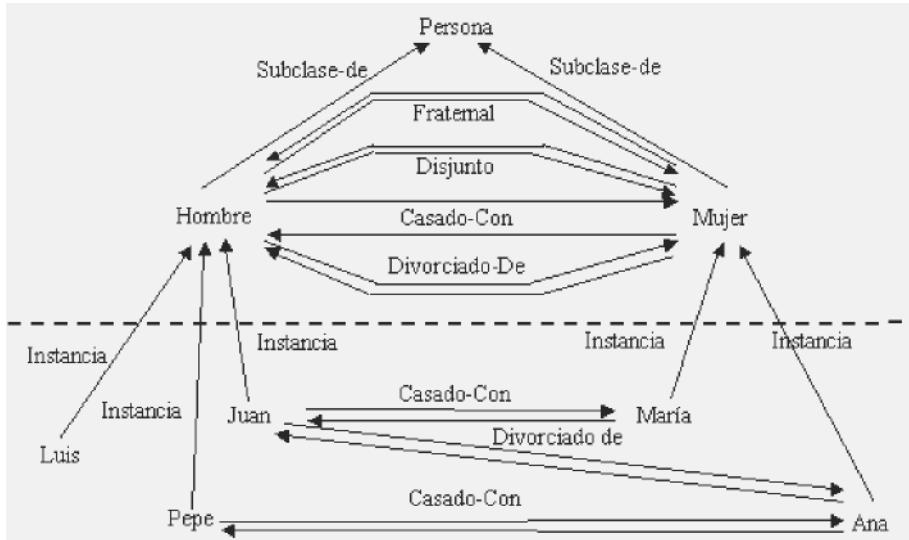


Figura 4.12: Ejemplo de una jerarquía en marcos.

4.4.1.2 Representación de relaciones entre conceptos

El formalismo de marcos representa las relaciones del dominio mediante relaciones entre marcos clase, entre marcos instancia, y entre marcos clase y marcos instancia, formando así un sistema basado en marcos (SBM). Intuitivamente, el significado del SBM de la Figura 4.12 es el siguiente:

- Los marcos clase *Hombre* y *Mujer* son subclases del marco clase *Persona*, y el marco clase *Persona* es una superclase de los marcos clase *Hombre* y *Mujer*.
- Los marcos instancia *Luis*, *Pepe* y *Juan* son instancias del marco clase *Hombre* y, por lo tanto, el conjunto de los *Hombres* está formado por *Luis*, *Pepe* y *Juan*.
- Los marcos instancia *Ana* y *María* son instancias del marco clase *Mujer* y, por lo tanto, el conjunto de las *Mujeres* está formado por *María* y por *Ana*.
- Los marcos clase *Hombre* y *Mujer* son hermanos, pues tienen el mismo padre.
- Los marcos clase *Hombre* y *Mujer* son disjuntos, pues no hay instancias que pertenezcan simultáneamente a ambos marcos clase.
- Las relaciones *casado-con* y *divorciado-de* definidas entre marcos clase representan que los *Hombres* y las *Mujeres* se casan y se divorcian. Las relaciones *casado-con* y *divorciado-de*, definidas entre marcos instancia, representan que *Juan* está casado con *María* y divorciado de *Ana*, y que *Ana* está casada con *Pepe*.

Al igual que ocurría en el formalismo de las redes semánticas, hay ciertas relaciones (*subclase-de* e *instancia*) que son independientes del dominio, y que reciben el nombre de **relaciones estándar**. La técnica de inferencia basada en herencia de propiedades razona sobre dichas relaciones. Sin embargo, también existen otras relaciones llamadas **relaciones no estándar**, para representar relaciones “a medida” entre conceptos de un dominio, sobre las que no se puede aplicar herencia de propiedades. La Figura 4.13 representa gráficamente la sintaxis de las relaciones más utilizadas en el paradigma de marcos, y que son las siguientes:

- *Subclase-de*, y su relación inversa *superclase-de*.
- *Instancia*, y su relación inversa *representa*.
- *Fraterno*.
- *Disjunto*.
- *No disjunto*.
- *Ad-hoc*, o relaciones “*a medida*”.

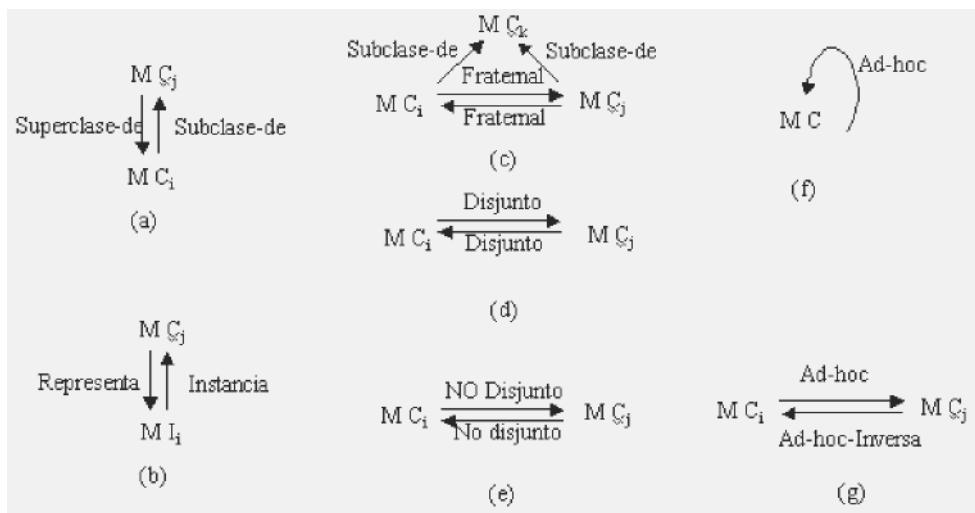


Figura 4.13: Sintaxis de relaciones en marcos.

Se consideran **relaciones estándar** las de *subclase-de* e *instancia*, y sus respectivas relaciones inversas llamadas *superclase-de* y *representa*. La relación *subclase-de* permite construir un SBM mediante la especialización de conceptos generales en conceptos más específicos. Sintácticamente, la relación *subclase-de* está bien definida si los marcos origen y destino son marcos clase. Desde el punto de vista semántico, el IC debe tener cuidado con que el significado de lo que quiere representar utilizando

estas relaciones sea correcto. La semántica correcta es que el concepto representado por MC_i es especialización o subconjunto del concepto representado por MC_j . La relación inversa de la relación *subclase-de* es la relación *superclase-de*. Dado que un concepto puede especializarse en varios conceptos o ser clasificado desde varios puntos de vista, a un marco clase pueden llegar y/o partir un número indefinido de relaciones *subclase-de* y *superclase-de*. Las relaciones *subclase-de* en los SBM definen un camino para la herencia de propiedades. Dada la jerarquía de la Figura 4.12, la herencia de propiedades permitirá que todas las propiedades que se definen en el marco clase *Persona* sean accesibles desde *Hombre* y desde todas sus instancias, definiéndose en *Hombre* sólo aquellas propiedades que distinguen a esta clase de la clase *Persona*, y en la clase *Persona* aquellas propiedades que son comunes a los marcos clase *Hombre* y *Mujer*.

Sintácticamente, una relación *instancia* está bien definida si tiene como origen un marco instancia MI_i y como destino un marco clase MC_j . Cuando el IC utiliza la relación *instancia* debe estar seguro de que el padre es un marco clase y de que el hijo es un marco instancia. Si no es así, sintácticamente la relación no tiene sentido, y la base de conocimiento (BC) será sintácticamente incorrecta. Semánticamente, esta relación representa que el marco instancia es un elemento del conjunto o clase representado por el marco clase. Dado que un elemento puede pertenecer a varios conjuntos simultáneamente, de un marco instancia pueden partir tantas relaciones *instancia* como conceptos lo describan consistentemente. Como se ha mencionado anteriormente, estas relaciones estándar permiten construir jerarquías o clasificaciones de conocimientos estáticos del dominio que se quiere modelar. Esta jerarquía será un árbol si existe un único camino que une cada marco instancia con el nodo raíz de la jerarquía, y un grafo en caso contrario.

Las relaciones **no estándar** representan dependencias entre conceptos de un dominio. El IC debe tener cuidado con que el significado de lo que quiere representar sea correcto. Como ejemplo de relaciones que representan la estructura de los conceptos del dominio se tienen las siguientes: *fraternal*, *disjunto* y *no-disjunto*.

- La relación *fraternal* está sintácticamente bien definida si los marcos origen y destino son marcos clase, y ambos están unidos mediante relaciones *subclase-de* con el mismo marco clase padre. Semánticamente, representa que dos conceptos son hermanos y, por consiguiente, tienen que tener el mismo padre.
- Una relación *disjunto* está sintácticamente bien definida si los marcos origen y destino son marcos clase. Semánticamente, representa que las clases son disjuntas, es decir, que tienen como intersección el conjunto vacío o, lo que es lo mismo, que los conjuntos o clases representados por ambos marcos no tienen ningún elemento en común. Además, esta relación está bien definida si no se ha definido previamente una relación *no-disjunto* entre ambos marcos clase.
- La relación *no-disjunto* es opuesta semánticamente a la relación *disjunto*. Por consiguiente, marcos clase conectados por relaciones *no-disjunto* pueden tener instancias comunes.

Aparte de estas relaciones no estándares, también podemos definir relaciones *a medida* o *ad-hoc*, como por ejemplo, *pertenece*, *casado-con*, *divorciado-de*, *conectado-a*, etc. Las relaciones *ad-hoc* expresan dependencias a medida entre conceptos de un dominio. Cuando se definen relaciones *ad-hoc* en un dominio, lo que ocurre frecuentemente, hay que asegurarse de definirlas entre marcos clase, y no entre marcos instancias.

Gráficamente, las figuras (f) y (g) en la Figura 4.13 muestran la sintaxis de relaciones *ad-hoc* entre marcos clase. Ejemplos de relaciones ad-hoc en la Figura 4.14 son las siguientes: la relación *casado-con* entre los marcos *Hombre* y *Mujer*; la relación *conectado-a*, que conecta, por ejemplo, computadoras en una red; y, las relaciones *compuesto-de* y *forma-parte-de*, utilizadas para describir un sistema especificando los subsistemas que lo componen y los sistemas de los que él forma parte. Cuando se definen relaciones *ad-hoc* entre dos marcos instancia, previamente hay que comprobar lo siguiente:

- Que la relación *ad-hoc* haya sido definida previamente entre dos marcos clase.
- Que los marcos instancia que se quiere conectar sean instancias de dichos marcos clase.

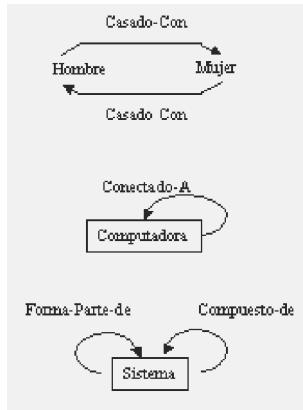


Figura 4.14: Ejemplos de relaciones *ad-hoc* entre marcos clases.

En el ejemplo de la Figura 4.12, antes de introducir la relación *casado-con* entre *Juan* y *María*, y entre *María* y *Juan*, el IC debe comprobar que la relación *casado-con* está definida en la BC y que *Juan* es instancia de *Hombre* y que *María* es instancia de *Mujer*. Solamente entonces, la relación se crea entre *Juan* y *María*, y entre *María* y *Juan*. Se procede de la misma forma para representar el hecho de que “*Ana está casada con Pepe*”. Es importante mencionar que cada relación *ad-hoc* entre dos marcos instancia es una instancia de la relación *ad-hoc* definida a nivel de clase. Por ejemplo, en la jerarquía de la Figura 4.12 existen cuatro instancias de la relación *ad-hoc casado-con*.

Las relaciones *ad-hoc* también se utilizan para conectar marcos clase de diferentes jerarquías. Normalmente, el IC construye varias jerarquías para representar clasificaciones de los principales conceptos que forman el sistema. Por ejemplo, supongamos un sistema basado en el conocimiento (SBC) para la liga de fútbol, como mínimo se necesitan dos jerarquías: una para clasificar a los jugadores de fútbol y otra para clasificar los equipos de fútbol en función de la división en la que juegan. Se puede, por tanto, definir una relación a medida *pertenecce* entre los marcos clase *Jugadores-de-fútbol* y *Equipos-de-fútbol* sujeta a las restricciones de que ambos, el *Jugador* y el *equipo*, tiene que pertenecer a la misma división, y que un *Jugador* pertenece a un único *equipo*. Cuando se crea una relación *pertenecce* instanciada entre un *Jugador* y un *equipo* concreto, además de comprobar que la relación está definida entre dos marcos clase y que dichas instancias lo son de dichos marcos clase, el IC, para evitar errores, comprobará que se cumplen las restricciones establecidas cuando se definió la relación. Gráficamente, la relación entre ambas jerarquías se muestra en la Figura 4.15.

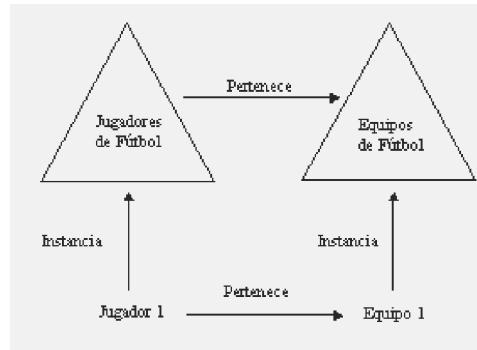


Figura 4.15: Ejemplos de relaciones *ad-hoc* entre diferentes jerarquías.

Muchas de las herramientas existentes para construir SBM no implementan este tipo de relaciones no estándar. Esto significa que el IC debe buscarse un “truco” para representarlas (por ejemplo, mediante otro marco), si realmente las necesita en su sistema. En este caso, a diferencia de las relaciones estándar, el motor de inferencias no trabajará con ellas, y el IC deberá implementar las inferencias bien con procedimientos programados, con reglas, con demonios, o con cualquier otro modo de expresar conocimientos procedimentales.

4.4.1.3 Representación de las propiedades de los conceptos

El IC utiliza dos tipos de propiedades cuando formaliza su BC en marcos: **propiedades de clase** y **propiedades de instancia**:

- Las *propiedades de clase* representan atributos o características genéricas de un concepto o clase. El IC define y rellena estas propiedades en el marco clase, y toman siempre el mismo valor en todos los elementos o instancias de la clase.

En la jerarquía de la Figura 4.16, la propiedad *animal-racional* en el marco clase *Persona* y la propiedad *sexo* en los marcos clase *Hombre* y *Mujer* son *propiedades de clase*.

- Las *propiedades de instancia*, aunque el IC las define en el marco clase y son comunes a todas las instancias del marco clase, se rellenan en cada instancia con valores concretos que dependen del elemento de la clase que se esté representando. Gráficamente, si la propiedad va precedida del símbolo “*” se trata de una propiedad de instancia. En el SBM de la Figura 4.16, las propiedades *nombre*, *edad*, *nacionalidad*, *estado-civil*, *tipo-de-matrimonio*, *religión*, *fecha-de-boda* y *fecha-de-nacimiento* son propiedades de instancia en el marco clase *Persona*.

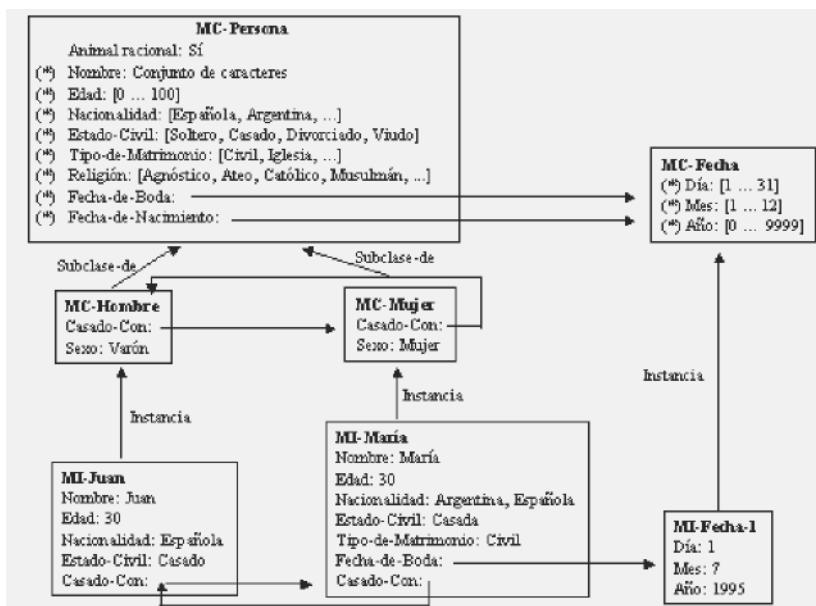


Figura 4.16: Ejemplos de propiedades en un SBM.

En la Figura 4.16, en los marcos clase, las propiedades de clase se han llenado con el valor que toma la propiedad, y las propiedades de instancia con el tipo de valor con el que éstas se pueden llenar en las instancias. Concretamente, con un tipo de datos (entero, conjunto de caracteres, etc.) o un puntero a un marco clase. Además, en los marcos instancia no se han llenado todas las propiedades de instancia definidas en los marcos clase. Esto es debido bien al desconocimiento del valor de una propiedad o bien a la intención de no repetir información. Por ejemplo, si se sabe que “Juan está casado con María”, los dos tendrán el mismo *tipo-de-matrimonio*, *fecha-de-boda*, etc. Este tipo de inferencias van asociadas a la relación *ad-hoc casado-con*. Dado que la mayoría de las herramientas no trabajan con este tipo de relaciones, el IC deberá crear un procedimiento para que el motor de inferencias sea capaz de extraer las mismas.

conclusiones que un humano en las mismas circunstancias. Al definir las propiedades de clase y de instancia en los marcos clase, el IC debe tener en cuenta lo siguiente:

- La distribución de las propiedades en el SBM debe favorecer que unos marcos compartan propiedades con otros marcos, evitando la presencia de conocimientos redundantes. Por ejemplo, la definición de la propiedad *nombre* o *animal-racional* en los marcos *Hombre* y *Mujer* haría que la jerarquía de la Figura 4.16 fuese redundante.
- Las propiedades deben proporcionar la suficiente información como para determinar la entidad que el marco clase está representando, discriminando las entidades representadas por él de las que no lo son. Como ejemplo, se tiene la propiedad *sexo* en los marcos clase *Hombre* y *Mujer*; si no estuviera definida haría indiscriminables las instancias de una y otra clase.
- El nombre de cada propiedad refleja su semántica, pero no indica la importancia que tiene la propiedad en el marco clase.
- El carácter local de cada propiedad permite tener propiedades con el mismo nombre en diferentes marcos. Por ejemplo, la propiedad *sexo* en los marcos clase *Hombre* y *Mujer*.
- Un marco clase rellena las propiedades de clase con unos valores concretos.
- En un marco clase se puede definir una propiedad de instancia utilizando otros marcos clase. Por ejemplo, las propiedades *fecha-de-nacimiento* y *fecha-de-boda* se definen a través del marco clase *Fecha*.
- Un marco instancia puede llenar o no todas las propiedades de instancia definidas en los marcos clase con los que está conectado. Por ejemplo, en el marco instancia *Juan* no se ha llenado las propiedades *fecha-de-boda* y *tipo-de-matrimonio* porque toman los mismos valores que los del marco instancia *María*. Otras propiedades, como la propiedad *religión* podrían no llenarse, por ejemplo, por desconocimiento del valor.

4.4.1.4 Representación de facetas de propiedades

Identificadas las relaciones y distribuidas las propiedades en los marcos clase, el IC pasa a describir las facetas de cada una de las propiedades. Las **facetas** permiten modelar características de las propiedades y relaciones en los marcos clase. Declarativamente, una propiedad se define especificando un puntero o un tipo de datos (lógico, carácter, número, entero, real, o como un conjunto restringido de valores numéricos, caracteres, etc.); proceduralmente, se define utilizando un procedimiento o una regla. Las facetas, independientemente de que se llenen con valores, punteros, o procedimientos, se clasifican en las siguientes tres categorías: facetas que definen propiedades de clase, de instancia y relaciones; facetas que definen propiedades de clase y relaciones; y facetas que definen propiedades de instancia. El motor de inferencias usa las facetas para mantener la integridad semántica de los datos, es decir, para

comprobar que los valores introducidos en las propiedades realmente pertenecen al tipo especificado, en la obtención de valores, y en la asignación de valores por defecto.

En cualquier propiedad o relación que defina el IC, obligatoriamente, debe especificar las **facetas que definen propiedades de clase, de instancia y relaciones**: *tipo ranura*, *cardinalidad mínima* y *cardinalidad máxima* de valores que puede tomar la propiedad o relación, y *multivaluada* si la propiedad o relación toma más de un valor.

- *Tipo ranura*: esta faceta establece el tipo de datos con el que se rellenará la propiedad o relación, garantizando que, una vez rellenada la propiedad o relación con unos valores concretos, dichos valores pertenecerán al tipo especificado en ella.
 - Si se trata de propiedades de clase o de instancia que se llenan con unos valores pertenecientes a determinados tipos de datos, en esta faceta se especificará el tipo correspondiente. Por ejemplo, en el marco clase *Persona*, las propiedades *nombre*, *estado-civil*, *religión*, *nacionalidad* y *tipo-de-matrimonio* se definen como un conjunto de caracteres; la propiedad *edad* se define como un entero corto; y, la propiedad *animal-racional* como de tipo lógico.
 - Si se trata de propiedades de clase o de instancia definidas como marcos, en esta faceta se especifica, precisamente, que se trata de un marco. Por ejemplo, la definición de las propiedades *fecha-de-nacimiento* y *fecha-de-boda* como un marco evitará repetir la descripción del concepto *Fecha* en cada una de ellas.
 - Si se trata de relaciones, la relación se definirá siempre en el marco clase origen de la relación, tendrá como nombre el de la relación, y en esta faceta se deberá especificar que se trata de un marco. Como ejemplo se tiene las relaciones *subclase-de*, *casado-con* y *divorciado-de* en los marcos clase *Hombre* y *Mujer*.
- *Cardinalidad mínima*: esta faceta establece el número mínimo de valores con los que se rellena la ranura, siempre que ésta se rellene. Así, la cardinalidad mínima de todas las ranuras definidas en el marco clase *Persona* es uno.
- *Cardinalidad máxima*: esta faceta informa del número máximo de valores con los que se puede llenar la ranura. Por ejemplo, en el marco clase *Persona*, la cardinalidad máxima de todas las propiedades es uno, salvo las cardinalidades de las propiedades *nacionalidad* y *tipo-de-matrimonio*. La propiedad *nacionalidad* podría llegar a tomar N valores, suponiendo que N sea el número máximo de nacionalidades que una persona puede tener.
- *Multivaluada*: esta faceta establece si la propiedad puede tener más de un valor o no. Si la *cardinalidad mínima* es igual a la *máxima*, y ambas son iguales a uno, entonces la propiedad no es multivaluada. En caso contrario, si la *cardinalidad*

mínima y la *máxima* son iguales y ambas diferentes de uno, o bien si la *cardinalidad mínima* es diferente de la *máxima*, entonces la ranura sí es *multivaluada*. El IC deberá realizar estas comprobaciones para evitar inconsistencias. En el ejemplo de *Persona*, las únicas propiedades multivaluadas son las de *nacionalidad* y *tipo-de-matrimonio*. El resto de las propiedades no son multivaluadas; y se pueden denominar simples.

Las propiedades de clase que se definieron en la faceta *tipo ranura* como un tipo de datos, rellenan la faceta **propiedad general**, que es una **faceta común a las propiedades de clase y relaciones**, con los valores que toma la propiedad en el marco clase. Dichos valores serán consistentes con los tipos especificados en la faceta *tipo ranura*. Las propiedades de clase definidas como marcos y relaciones rellenan esta faceta con un puntero a un marco clase. Las propiedades de instancia nunca rellenan esta faceta con valores o punteros. Conviene, no obstante, asignar a las propiedades de instancia un cierto valor que indique que no se rellenan; por ejemplo, el símbolo “—”. Por ejemplo, en el marco clase *Persona* la propiedad de clase *animal-racional* rellena esta faceta con un valor de tipo lógico; la relación *superclase-de* con punteros a los marcos clase *Hombre* y *Mujer*; y, las propiedades de instancia con el símbolo “—”.

Para cada una de las propiedades de instancia definidas en un marco clase, se definirán las correspondientes **facetas que definen propiedades de instancia: valores permitidos de la propiedad, valores por omisión o por defecto** asignados a la propiedad, *si necesito, si modifco, si añado y si borro*, que se utilizan al necesitar, modificar, añadir o borrar un valor de una propiedad.

- *Valores Permitidos.* Esta faceta especifica el conjunto de valores válidos que puede tomar la propiedad de instancia. Concretamente, se utilizará: un tipo de datos, un rango de valores, o un puntero a un marco clase. Independientemente de cómo se defina la propiedad, el IC debe comprobar que el conjunto de valores especificados es consistente con el tipo almacenado en la faceta *tipo ranura*. Ejemplos de definiciones para el marco clase *Persona* son los siguientes:

- Las propiedades *nombre, nacionalidad* y *tipo-de-matrimonio* tienen como valores permitidos un tipo de datos.
- Los *rangos de valores* se utilizan para restringir el número de posibles valores con los que se rellena la propiedad en la instancia. Con el fin de preservar la integridad semántica de la BC, el rango de valores impide que propiedades de instancia se llenen con valores que sí satisfacen la definición dada en la faceta *tipo ranura*, pero que no se dan en el dominio de la aplicación. Aunque ello conlleve un mayor esfuerzo, se aconseja al IC que defina un rango de valores frente a un tipo de datos. Ejemplos del uso de un rango de valores son las propiedades *edad, estado-civil* y *religión*.
- Las propiedades de instancia definidas en la faceta *tipo ranura* como marcos rellenan esta faceta con un puntero al marco clase que define la propiedad. Como ejemplo, se tienen las propiedades *fecha-de-nacimiento* y *fecha-de-boda* en el marco clase *Persona*.

- *Valores por Omisión.* Esta faceta define los valores que debe tomar la propiedad de instancia en un marco instancia si no se conoce de forma explícita otro valor. No obstante, marcos instancia y marcos clase pueden anular el valor por omisión dado a la propiedad en otro marco clase, bien al asignar un valor nuevo, como ocurre, por ejemplo, con las propiedades que presentan excepciones, bien al añadirle valores adicionales. Es aconsejable, como siempre, llenar las propiedades de instancia que no tienen asociados valores por omisión, las propiedades de clase y las relaciones con el símbolo “—”. El valor asignado en esta faceta será siempre del tipo especificado en la faceta *tipo ranura*. Por ejemplo, si la jerarquía de *Personas* se va a utilizar únicamente en *España*, por omisión, la propiedad *nacionalidad* se puede llenar con el valor *española*.
- *Si Necesito.* Esta faceta almacena un procedimiento o regla que se ejecuta al solicitar el valor de una propiedad de instancia en un marco instancia y ser desconocido dicho valor. Estos procedimientos o reglas se utilizan para:
 - *Requerir el valor de una propiedad.* Si el valor de una propiedad de instancia en un marco instancia es desconocido, los procedimientos pueden preguntar dicho valor al usuario del sistema, o bien, pueden calcular el valor a partir de otros valores conocidos almacenados en la BC.
 - *Mantener la integridad semántica de la BC.* Cada vez que el usuario introduzca un valor en alguna propiedad de una instancia, el SBM comprobará que se cumplen las restricciones impuestas en las facetas de dichas propiedades, y sólo si los valores son correctos se aceptan. Nótese que ésta es una funcionalidad extremadamente interesante del formalismo de marcos, puesto que la propia estructura de los marcos permite mantener las restricciones de integridad semántica entre los elementos del dominio. El IC deberá aprovechar esta funcionalidad ofrecida por el formalismo para disminuir las posibilidades de valores erróneos en los contenidos de las propiedades llenadas en los marcos instancia.
 - *Gestionar dinámicamente de valores.* Los procedimientos almacenados en la faceta *si necesito* también se utilizan para obtener dinámicamente el valor de una propiedad de un marco instancia si el conjunto de valores con los que se rellena la propiedad en la instancia se puede obtener a partir de los valores almacenados en otras propiedades y relaciones.
 - *Determinar dinámicamente el rango de valores.* Si el conjunto de valores con los que se rellena una propiedad en el marco instancia dependen de los valores que se han llenado en otras propiedades y relaciones del mismo o de otros marcos instancia, dinámicamente se puede determinar el rango de valores permitidos definiendo un procedimiento en esta faceta.
- *Si Modifico.* Esta faceta almacena el procedimiento que se ejecuta al modificar un valor de una propiedad de un marco instancia. La ejecución de este procedimiento puede añadir, modificar y borrar valores de otras ranuras, disparando así sus procedimientos asociados. Por ejemplo, si se modifica el *estado-civil* de

Juan, pasando a estar *divorciado*, al modificar este valor, se deberían borrar del SBM las relaciones *ad-hoc casado-con* entre los marcos instancia *Juan* y *Maria*, y se deberían crear dos nuevas relaciones *ad-hoc divorciado-de* entre dichos marcos instancia.

- *Si Añado*. En esta faceta se almacenan procedimientos que se ejecutan al introducir un valor en una propiedad de un marco instancia que estaba vacía. Se utiliza para:(a) Mantener la integridad semántica de la BC; y (b).añadir, modificar y borrar valores de otras ranuras.
- *Si Borro*. Esta faceta almacena el procedimiento que se ejecuta al borrar un valor en una propiedad de un marco instancia. La ejecución de este procedimiento puede añadir, modificar y borrar valores en otras ranuras, disparando de este modo sus procedimientos asociados.

4.4.2 Criterios de diseño

En este apartado se presentan algunos criterios de diseño que pueden ayudar al IC a representar el conocimiento utilizando el formalismo de marcos:

- Se debe favorecer la compartición de propiedades de clase y de instancia entre marcos.
- Se debe evitar representar conocimientos redundantes.
- En cada marco clase debe haber una propiedad de clase que identifique a los elementos de dicha clase.
- Semántica de las propiedades: las propiedades deben tener o aportar significado a la representación.
- Debido al carácter local de las propiedades, se pueden tener propiedades repetidas con el mismo nombre en diferentes marcos clase.

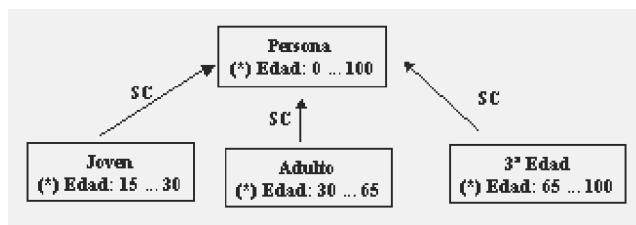


Figura 4.17: Ejemplo de redefinición de propiedades.

- En caso necesario, se pueden redefinir las propiedades (de clase e instancia) en marcos clase más específicos (tal y como muestra el ejemplo de la Figura 4.17).

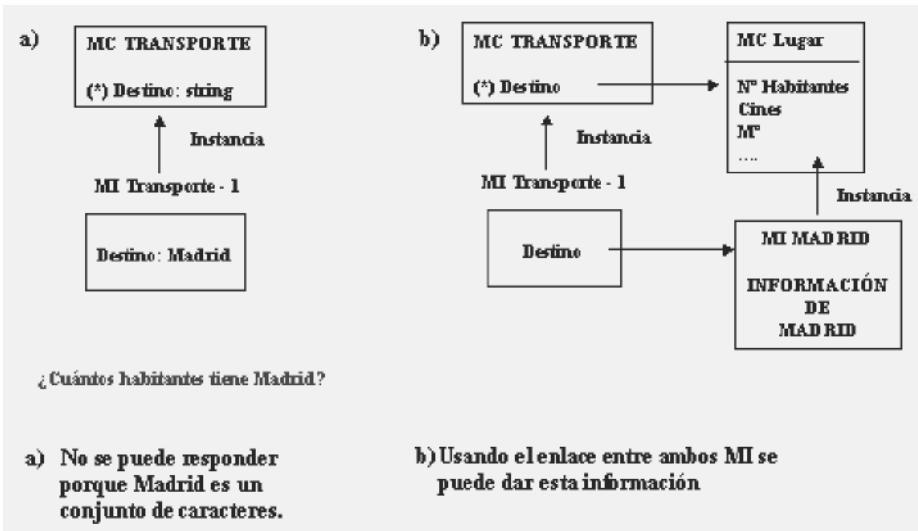


Figura 4.18: Ejemplo de propiedad definida como marco.

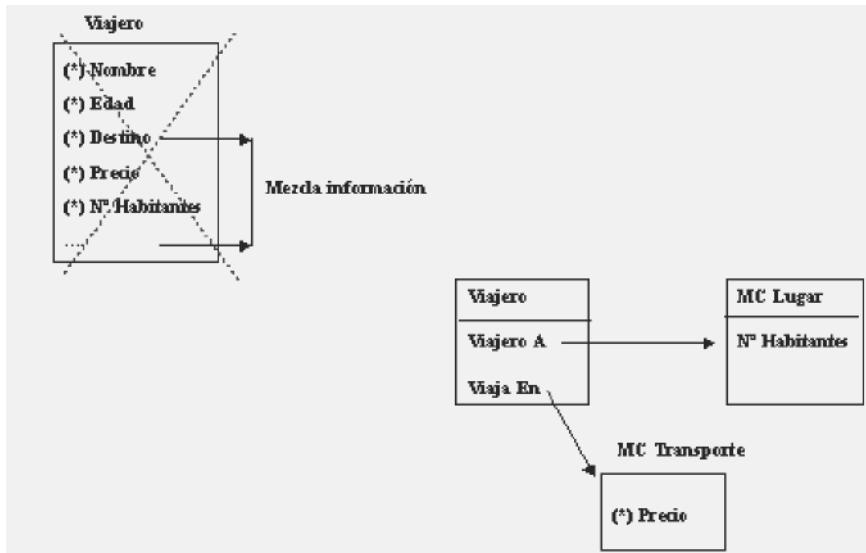


Figura 4.19: Ejemplo de representación de varios conceptos.

- Si se desea conocer información o propiedades de las propiedades de un marco, entonces dichas propiedades deben ser definidas como marcos (tal y como se muestra en el ejemplo de la Figura 4.18).

- No se deben mezclar conceptos. Por ejemplo: el caso de un viajero que va a un destino usando un transporte que tiene un precio se debería representar como se muestra en la Figura 4.19.
- Hay que tener en cuenta que en un marco instancia se pueden llenar, o no, todas las propiedades de instancia definidas en los marcos clase con los que está conectado.
- Hay que recordar que las propiedades de instancia se llenan con un valor concreto.
- Es importante recordar que no se pueden utilizar en las instancias propiedades que no se hayan definido en los marcos clase.

4.5 Inferencia de conocimiento en SBM

El formalismo de marcos permite realizar inferencias utilizando los conocimientos almacenados en la BC. En concreto, las tres técnicas que utilizan los marcos, de forma integrada o de manera individualizada, son: la **equiparación** para clasificar entidades en la BC; la **herencia** de propiedades para compartir propiedades entre marcos; y los **valores activos** para representar la conducta del sistema y mantener la integridad de los datos almacenados.

4.5.1 Equiparación

Equiparar significa clasificar. En este sentido, conocidos los valores de un conjunto de propiedades que describen parcialmente una nueva entidad o marco pregunta, la técnica de equiparación clasifica el marco pregunta en el árbol o en el grafo que representa los conceptos del dominio. Para ello, la técnica de equiparación tiene que encontrar los marcos clase (uno o varios) de la BC que describen más consistentemente al marco pregunta. Encontrados los marcos clase (uno o varios), el marco pregunta se convierte en un marco instancia de dichos marcos clase. El problema es el siguiente: si los valores de todas las propiedades de una entidad son conocidos y dichas propiedades se encuentran almacenadas en un único marco clase de la BC, entonces la equiparación de la nueva entidad con el marco clase es perfecta. Sin embargo, esta situación raramente ocurre debido a que:

- Las propiedades conocidas en la nueva entidad se encuentran distribuidas en la BC por todos los marcos clase del árbol o grafo de conceptos.
- No todas las propiedades de la nueva entidad son conocidas al empezar la equiparación. Los conocimientos que se poseen de la entidad son limitados, pues, habitualmente, se conoce un conjunto finito, no muy amplio, de sus propiedades.
- Aun conociendo los valores de algunas propiedades de la nueva entidad, normalmente se tienen distintos grados de seguridad en los valores que se han asignado.

- No todas las propiedades almacenadas en el marco pregunta aportan la misma información a la hora de realizar la equiparación.
- Los valores que por omisión se han asignado a las propiedades en los marcos clase no siempre representan información cierta para todas las instancias, pues existen excepciones para dichos valores. Se pueden realizar varias equiparaciones correctas de un marco pregunta con varios marcos clase.

Básicamente, las tres etapas en las que se descompone la técnica de equiparación son: *selección de marcos candidatos*, *cálculo de valores de equiparación* y, finalmente, *decisión*.

1. La *selección de marcos candidatos* se realizará siguiendo alguno de los siguientes procedimientos:
 - Si el *tipo* de una nueva entidad es conocido, se puede seleccionar el marco clase en el que se ha definido el *tipo*, y todos los marcos clase en los que éste se ha especializado. Por ejemplo, si se sabe que se trata de un perro, entonces se selecciona el marco clase *Perro* y todas sus subclases, pero no el marco clase *Mamífero* o el marco clase *Gato*.
 - Si el *tipo* de la nueva entidad es desconocido, la selección de marcos clase se realiza arbitrariamente, o se eligen aquellos marcos clase en los que, como mínimo, se encuentre definida una propiedad conocida en el marco pregunta. En este caso, los marcos clase que no tengan al menos una propiedad en común con el marco pregunta no se seleccionan. Por ejemplo, dada una entidad que tiene pelo y cuatro patas, se utilizarían las propiedades *Tiene-Pelo* y *Número-Patas* para la selección de marcos candidatos.
2. Seleccionados los marcos clase candidatos, se pasa a *calcular el valor de equiparación* (VE) de la nueva entidad en cada una de las clases seleccionadas. El VE es una medida que informa del grado de idoneidad de la equiparación que se va a realizar. El método de cálculo del VE variará de unas aplicaciones a otras. Por ejemplo, si los valores de la propiedad en la nueva entidad no coinciden o no están incluidos en los valores almacenados en la clase, y estas propiedades son esenciales, entonces el VE de la nueva entidad con la clase puede ser cero. Pero si las propiedades no son esenciales, la opción podría ser disminuir el VE. En caso de que los valores de la propiedad en la entidad coincidan con los valores de la propiedad en la clase, se aumentará proporcionalmente el VE, según sea de esencial la propiedad en la clase. El IC, según sean las características de la aplicación, elaborará un fórmula para el cálculo del VE.
3. Calculados los VE, la técnica de equiparación *decide con qué marcos clase se equipará la nueva entidad*. Si el VE es lo suficientemente alto y el marco es lo suficientemente específico, entonces el sistema no buscará otros marcos e instanciará la nueva entidad convirtiéndola en un marco instancia. Sin embargo, si el VE no es lo suficientemente alto, o el marco no es lo suficientemente específico,

entonces la técnica tendrá que identificar marcos relevantes. Esto puede realizarse de varias maneras, entre las que cabe destacar las siguientes: ascender a lo largo de la jerarquía hasta encontrar una equiparación relevante; comenzar en el marco raíz y seleccionar el marco con el mejor VE; buscar marcos relacionados utilizando las relaciones *Fraterno*, *Disjunto*, *No-Disjunto* y, o relaciones *ad-hoc*.

Esta técnica es especialmente útil en SBC que clasifican o en sistemas que se enfrentan a situaciones parecidas a otras que ocurrieron anteriormente. Por ejemplo, la técnica de equiparación se podría utilizar en un sistema que clasifique bacterias concretas en una taxonomía de bacterias, o en un sistema médico ante el cuadro de un paciente concreto, o en sistemas industriales ante fallos particulares, etc.

4.5.2 Herencia de propiedades

Esta técnica permite compartir valores y definiciones de propiedades entre marcos de una BC usando las relaciones *instancia* y *subclase-de*. Se puede diferenciar entre **herencia simple**, que se aplica a BC en forma de árbol, **herencia múltiple**, que se aplica a BC en forma de grafo.

- *Herencia simple*. Se aplica herencia simple a una BC formalizada en marcos cuando sólo existe un único camino que une el marco instancia con el nodo raíz de la jerarquía; es decir, cuando cada marco instancia es instancia de un único marco clase y cuando cada marco clase es, a su vez, especialización de una única clase. El algoritmo que realiza la inferencia para encontrar los valores de una cierta propiedad de un marco instancia es el siguiente:

1. Se busca la propiedad en el marco instancia. Si se encuentra, se devuelven sus valores. En caso contrario, se accede al marco clase padre utilizando la relación *instancia*.
2. Se busca la propiedad en el marco clase. Si se encuentra la propiedad, entonces se devuelven sus valores y el algoritmo finaliza. En otro caso, se utiliza la relación *subclase-de* para acceder al marco clase padre. Este paso se repite mientras el padre no sea el marco raíz del árbol.
3. Se busca la propiedad en el marco raíz. Si se encuentra, se devuelven sus valores. Si no, la técnica debe responder que con los conocimientos almacenados en la BC es imposible proporcionar una respuesta.

Si en el camino que une el marco instancia con el marco raíz se encuentran propiedades que presentan excepciones, la técnica de herencia simple proporcionará el valor de la propiedad situada en el marco clase más cercano a la instancia, al ser el orden en el que se recorre el árbol un orden total.

Si se solicita el valor de una propiedad de instancia que ya toma valor en el marco instancia, la técnica de herencia simple devuelve el valor local almacenado en él.

Si se solicita el valor de una propiedad de instancia que no tiene valores asignados en el marco instancia, entonces la técnica, utilizando las relaciones *instancia* y

subclase-de, buscará la propiedad en los marcos clase con los que la instancia se relaciona. La ejecución de un procedimiento asociado a la faceta Si Necesito, o la herencia de un valor por omisión, permitirán, en última instancia, responder a la pregunta. Nótese que en caso de estar definidas ambas facetas, debe existir una estrategia de control que ayude al SBM a decidir si ejecuta un procedimiento o si toma el valor por omisión. Si se solicita el valor de una propiedad de clase accesible desde un marco instancia, la técnica de herencia simple buscará la propiedad en el árbol y devolverá su valor.

- Se aplica *herencia múltiple* a una BC formalizada en marcos si existen varios caminos que unen los marcos instancia con el nodo raíz de la jerarquía, bien porque un marco instancia es instancia de varios marcos clase, o bien porque cada marco clase es especialización de una o varias clases. Dado que un hijo puede heredar propiedades de varios padres, la dificultad se encuentra en determinar el orden en el cual las clases se deben explorar. Si los padres tienen propiedades con distinto nombre nunca habrá conflicto. Sin embargo, si los padres tienen propiedades con el mismo nombre, el valor de la propiedad que hereda el marco hijo dependerá de la técnica de búsqueda que se utilice para recorrer el grafo. Por consiguiente, existen diferentes algoritmos que recorren de forma diferente el grafo, y el IC deberá elegir uno de ellos:
 - *Búsqueda en profundidad*. Esta técnica explora en profundidad todos los posibles caminos que van desde el marco instancia al marco raíz del SBM. Algunos de los criterios que, de forma combinada, pueden usarse para establecer el orden en el que se recorrerán cada uno de sus marcos clase son:
 1. Recorrer el grafo de *izquierda a derecha*.
 2. Usar el criterio de *exhaustividad* para evitar la búsqueda reiterada de propiedades en marcos clase ya analizados, es decir, solamente se buscará la propiedad en cada marco una única vez.
 3. El problema que presentan las técnicas de herencia múltiple, que recorren el grafo en profundidad de forma exhaustiva, y de izquierda a derecha, es que, para una propiedad que presenta excepciones, las instancias pueden heredar los valores de clases generales en vez de heredar los valores de clases más específicas. Esto ocurre siempre que el marco que presenta la excepción está situado en la parte derecha de la jerarquía. Con el fin de paliar este error, el criterio de *especificidad* impone la restricción de que sólo se puede buscar la propiedad en una clase si previamente se ha buscado en todas sus subclases.

Un algoritmo que cumple los tres requisitos anteriores es el *procedimiento de ordenación topológica*. Dada una instancia, este procedimiento utiliza la topología del grafo para formar su lista de criterios de preferencias. Esta lista almacena el orden en el que se recorrerán todos los marcos clase accesibles desde un marco instancia concreto.

- *Búsqueda en amplitud: Longitud del camino.* La técnica de longitud del camino recorre el grafo por niveles que están a igual distancia del marco instancia. Dado un marco instancia, se mirará si la propiedad se encuentra en alguno de los marcos clase con los que está unido. Si se encuentra, entonces se devuelve su valor. En caso contrario, utilizando las relaciones *subclase-de*, comienza a buscar la propiedad en todas las superclases de dichas clases. En otras palabras, primero se busca la propiedad en los padres, es decir, en aquellos marcos clase que están a distancia uno del marco instancia. Si no la encuentra, entonces la técnica busca en los abuelos, es decir, en aquellos marcos que están a distancia dos, y así sucesivamente. El proceso termina al encontrar la propiedad o al alcanzar el nodo raíz sin encontrarla. Este algoritmo es eficiente y es adecuado en sistemas de herencia que razonan sobre grafos sin propiedades repetidas. Si el grafo contiene propiedades con excepciones aparecerán dos problemas: razonar en presencia de ambigüedades y de conocimientos no especializados.

Un problema adicional que presenta la técnica de la longitud del camino es que la longitud de un camino puede no corresponder con el nivel de generalidad de una clase, si hay clases generales que no se han especializado con el mismo nivel de detalle que sus clases hermanas. El IC debe evitar en lo posible construir grafos que presenten diferente granularidad en la especialización de las subclases.

- *La distancia “inferencial”.* Para resolver algunos de los problemas anteriores, Touretzky [Touretzky, 1986] propuso el concepto de distancia “inferencial”. La distancia “inferencial” razona correctamente en BC redundantes, pero, aunque detecta situaciones ambiguas, no las resuelve. La distancia “inferencial” se puede definir como: “la condición necesaria y suficiente para que la clase₁ esté más cercana a la clase₂ que a la clase₃ es que la clase₁ tenga un camino de inferencia a través de la clase₂ hacia la clase₃; en otras palabras, si la clase₂ está entre la clase₁ y la clase₃”. Obsérvese que en el concepto de distancia “inferencial” aparecen marcos conectados por enlaces de tipo *subclase-de*. Por este motivo, al no permitir comparar marcos no conectados, el orden introducido por la distancia “inferencial” es un orden parcial y, por consiguiente, se pueden heredar valores contradictorios definidos en ramas que no están conectadas.

4.5.3 Valores activos

Además de la estructura declarativa de los marcos, también existe un aspecto dinámico o procedimental de los mismos. En muchos sistemas, estos procedimientos son el mecanismo principal para realizar razonamientos que no están cubiertos por las otras dos técnicas explicadas anteriormente. Con los nombres de *demonios*, *valores activos* o *disparadores*, se denomina a los procedimientos que recuperan, almacenan, y borran información en los SBM. Estas procedimientos, como se explicó anteriormente en este capítulo, se definen en las facetas *Si Necesito*, *Si Añado*, *Si Modifico*

y *Si Borro* de las propiedades de instancia de los marcos clase. La ejecución de estos procedimientos presenta las siguientes características:

- Los procedimientos se definen en el marco clase y permanecen latentes, sin hacer nada, a menos que, por algún motivo, se solicite su ejecución desde un marco instancia. Se puede solicitar la ejecución de un procedimiento para: recuperar, almacenar, o borrar valores de propiedades en marcos instancia; mantener la integridad semántica de la BC al impedir que se introduzcan valores en la BC que no satisfacen unas determinadas restricciones; garantizar que cambios en los valores de una propiedad se reflejan correcta y automáticamente en los valores de otras propiedades; calcular dinámicamente valores de propiedades que se requieren y que se obtienen a partir de los valores de otras propiedades almacenadas en la BC; y, finalmente, manejar errores.
- Los procedimientos se despiertan al recibir una solicitud de ejecución. Cuando un marco instancia solicita la ejecución de un valor activo, el procedimiento asociado se ejecuta con los valores almacenados en las propiedades del marco instancia.
- Los procedimientos pueden simular encadenamiento hacia adelante, y así tener demonios dirigidos por los eventos, y encadenamiento hacia atrás, y tener demonios dirigidos por las metas.
 - Los demonios dirigidos por los eventos ejecutan procedimientos antes de almacenar o borrar valores en las propiedades de un marco instancia, y están asociados a las facetas *Si Añado*, *Si Modifico* y *Si Borro*.
 - Los demonios dirigidos por las metas están asociados a la faceta *Si Necesito*, y deducen valores de propiedades a partir de valores almacenados en otras propiedades, utilizando un eficiente pero complejo mecanismo de encadenamiento hacia atrás.
- El control va pasando de unas propiedades a otras a medida que se van ejecutando los procedimientos. En cada instante, el control lo tiene la propiedad del procedimiento que se está ejecutando. Una vez ejecutado un procedimiento, el control vuelve al procedimiento que lo llamó, procedimiento que sigue ejecutándose en el mismo lugar en el que se detuvo.

4.6 Resumen

Dado que son múltiples los formalismos que permiten representar los conocimientos de un dominio, el objetivo de este libro no es realizar un estudio completo y exhaustivo de todos ellos. En la parte II del presente libro se han incluido los formalismos más relevantes explicando las nociones más importantes de cada uno de ellos. En concreto, en este capítulo 4 se han explicados los formalismos de redes semánticas y marcos. Finalmente, en este apartado se pretende resumir de una manera práctica

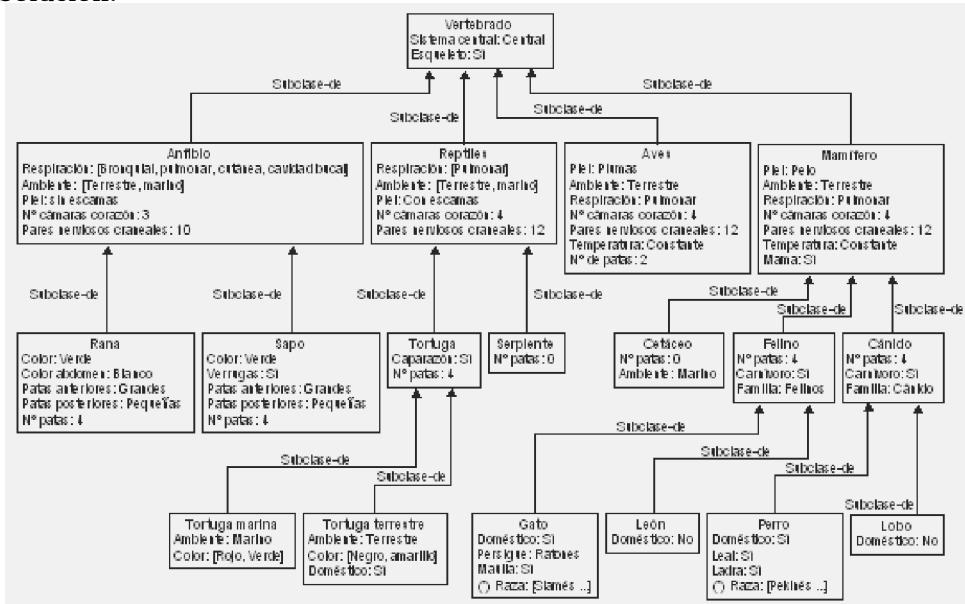
las ventajas e inconvenientes de los dos formalismos descritos en este capítulo: las redes semánticas y los marcos. En líneas generales, la mayoría de los SBC suelen representar los conocimientos del dominio utilizando marcos frente a redes semánticas debido a que:

- Los marcos permiten construir taxonomías de conceptos por especialización de conceptos generales en conceptos más específicos, lo que posibilita la herencia de propiedades. Con la herencia de propiedades se comparten propiedades entre marcos y se evita la presencia de conocimientos redundantes en la BC. Aunque esta característica es común a las redes semánticas, la diferencia se encuentra en que en los marcos las propiedades están recogidas dentro del marco, mientras que en las redes semánticas las propiedades no están agrupadas, es decir, se encuentran dispersas por toda la red uniendo nodos que representan conceptos con otros nodos que representan los valores que toma la propiedad.
- En las BC formalizadas en marcos las propiedades se pueden definir de forma declarativa y procedimental; sin embargo, en las redes semánticas sólo se pueden definir de forma declarativa. Además, la cantidad de información asociada a las propiedades en un marco supera con creces a la de las propiedades en una red semántica. En los marcos se puede introducir información sobre el tipo de datos al que pertenecen los valores, número de valores mínimo y máximo con los que se rellena cada propiedad, valores por omisión, excepciones, y procedimientos y reglas que permiten inferir los valores de dichas propiedades, mientras que en las redes semánticas no se puede.
- Los SBM permiten el uso de valores por omisión y excepciones a dichos valores; sin embargo, en las redes semánticas no se pueden representar valores por omisión, aunque sí excepciones. La propia estructura interna de los marcos permite mantener internamente las restricciones de integridad semántica que existen entre los elementos de un dominio, mientras que en las redes semánticas, no.
- Uno de los principales inconvenientes que presentan las redes semánticas es que a medida que la BC del sistema aumenta, la comprensión de la red se hace cada vez más difícil. No ocurre lo mismo en los SBM, los cuales facilitan el diseño y mantenimiento de la BC.

4.7 Ejercicios resueltos

4.1. Construir una jerarquía de marcos sobre animales vertebrados. La jerarquía debe estar formada al menos por diez marcos clase y, en cada uno, debe haber al menos dos propiedades de clase.

Solución:



4.8 Ejercicios propuestos

4.1. Suponer que un usuario que desea alquilar un coche por Internet rellenando el formulario que se presenta en la Figura 4.20. Los valores que toman los campos son los siguientes:

- Clase de coche: compacto, económico y lujo.
 - Tipo de coche: 2 puertas, 4 puertas y tracción a las cuatro ruedas.
 - Caja de cambios: manual, automática.
 - Aire acondicionado: sí, no.
 - Distancia: kilómetros o millas.

Suponer que el usuario desea que el punto de recogida y de entrega sea Madrid, que la clase de coche sea un compacto, de cuatro puertas y que la caja de cambios sea manual. Las salidas que obtiene del sistema son las mostradas en la Figura 4.21.

Se pide:

- Construir la red semántica que formaliza el dominio de alquiler de coches. Debe formalizarse: el punto de entrega y de recogida, la hora de entrega y de recogida, la fecha de entrega y recogida, la clase de coche, si tiene aire acondicionado, el tipo de caja de cambios, la tarifa diaria y el precio total estimado.

- Formalizar con redes semánticas una de las 5 respuestas que da el sistema y relacionar dichas respuestas con la formalización previamente realizada en el apartado anterior.

Punto de recogida: []

Enero 30 09:00

Punto de entrega: []

Febrero 3 09:00

Buscar

Agregue los siguientes datos sólo si desea acotar la búsqueda.

Clase de coche:

Tipo de coche:

Caja de cambio:

Aire acondicionado:

Compañía de alquiler de coches preferida: 1: 2:

Distancia en:

Mostrar tarifas con kilometraje: No Sí

Búsqueda avanzada

Figura 4.20: Formulario a llenar por el cliente.

Automóviles: Madrid, España						
Recogida en : Barajas (MAD), Madrid, España el lunes, 30 de enero 2006 09:00						
Entrega en : Barajas (MAD), Madrid, España el viernes, 03 de febrero 2006 09:00						
Compañía de alquiler de coches	Información	Diaría Tarifa	Precio total estimado	Kilómetros gratuitos	Cargo por kilómetro extra	Ubicación
Budget- ZD	Compacto, 4 puertas, Manual, Aire acondicionado	45.00 EURO *	180.00 EURO *	Ilimitado	-	Terminal
Hertz- IE	Compacto, 4 puertas, Manual, Aire acondicionado	27.62 EURO	191.38 EURO	Ilimitado	-	Terminal
Avis Rent a Car- IZ	Compacto, 4 puertas, Manual, Aire acondicionado	51.43 EURO *	205.74 EURO *	Ilimitado	-	Terminal
Alamo- AL	Compacto, 4 puertas, Manual, Aire acondicionado	45.80 EURO *	242.55 EURO *	Ilimitado	-	Terminal
Europcar- EP	Compacto, 4 puertas, Manual, Aire acondicionado	63.07 EURO	252.28 EURO	Ilimitado	-	Terminal

Figura 4.21: Coches disponibles.

4.2. Suponer que un usuario desea reservar un hotel por Internet rellenando el formulario de la Figura 4.22. Los valores que toman los campos son los siguientes:

- Provincia: son todas las provincias españolas.
- Población: se refiere a las poblaciones de las distintas provincias.
- Localización, que toma los valores: capital, capital y radio<25km.
- Ocupación, que toma los valores: 1 adulto, 2 adultos, 1 adulto y un niño, etc.
- Régimen: alojamiento, alojamiento y desayuno, media pensión y pensión completa.
- Categoría: de una a cinco estrellas.

The form is titled "Encuentre su hotel". It has several input fields and a date picker:

- País*: ESPAÑA
- Provincia*: MADRID
- Población: madrid
- Localización: Capital
- Hotel: (empty)
- Dates: Entrada: 23 Ago 2007. Calendar shows Agosto 2007 with 23 highlighted.
- Nº de noches: 1
- Nº Hab.: 1 Adulto
- Nº Hab.: 1 Adulto
- Nº Hab.: 1 Adulto
- Régimen: Aloj. y Desay.
- Categoría: ***

A "BUSCAR HOTELES" button is at the bottom.

Figura 4.22: Formulario a llenar por el cliente.

El usuario desea una habitación individual para un adulto en un hotel de al menos tres estrellas en Madrid capital para el día 23 de agosto de 2007 en régimen de alojamiento y desayuno. Las salidas que obtiene son las mostradas en la Figura 4.23.

Resultados de la Búsqueda:			
Se han encontrado 101 hoteles que cumplen sus parámetros de búsqueda:			
País: ESPAÑA Provincia: MADRID D.Obligació: madrid Localización: Capital Entrada: 23-09-2007 Salida: 24-09-2007 Habitaciones: 1 [1 Adulto] Régimen: Aloj. y Desay. Categoría: ***			
Hotel	Categoría	Población	Precio
<input checked="" type="checkbox"/> HOSTAL APOLÓ	***	MADRID	51,00 EU RESERVAR
<input checked="" type="checkbox"/> ARTURO SORIA SUITES	****	MADRID	59,99 EU RESERVAR
<input checked="" type="checkbox"/> CABALLERO ERRANTE	***	MADRID	69,99 EU RESERVAR
<input checked="" type="checkbox"/> LOS CONDES	***	MADRID	62,63 EU RESERVAR
<input checked="" type="checkbox"/> ARISTOS	***	MADRID	64,20 EU RESERVAR
<input checked="" type="checkbox"/> SILKEN TORRE GARDEN	***	MADRID	64,63 EU RESERVAR
<input checked="" type="checkbox"/> GRAN ATLANTA	***	MADRID	64,95 EU RESERVAR

Figura 4.23: Hoteles disponibles.

Se pide:

- Construir la red semántica para este dominio.
- Formalizar con redes semánticas una de las respuestas queda el sistema y relacionar dichas respuestas con la formalización previamente realizada en el apartado anterior.

4.3. Construir una BC formalizada en marcos que represente el Sistema Periódico (SP). Tras la etapa de conceptualización se sabe que:

- Los elementos del SP se clasifican en los siguientes grupos: Metales, No-Metales, Semi-Metales y los Gases Nobles, a excepción del Hidrógeno que no tiene un grupo bien definido. Además se sabe que, el Hidrógeno, los Metales, los No-Metales y los Semi-Metales reaccionan con otros elementos, mientras que los otros no.
- Los elementos Metales se pueden clasificar en dos grupos: Metales de No Transición, que a su vez engloba a los Alcalinos y Alcalinotérreos; y los Metales de Transición, que comprenden la primera y segunda serie de transición y a los Lantánidos y Actínidos, también llamados primera y segunda serie de transición interna.
- La clasificación de los No-Metales no está tan bien definida como en la clase de los Metales. Sin embargo, existe un grupo bien diferenciado que es el de los Halógenos.
- Los elementos del SP sólo pertenecen a un único grupo. A saber:
 - Los Alcalinos son el grupo Ia (Li, Na, K, Rb, Cs, Fr).
 - Los Alcalino-Térreos son el grupo IIa (Be, Mg, Ca, Sr, Ba, Ra).
 - La primera serie de transición interna son los grupos de los Lantánidos (Ce, Pr, Nd, Pm, Sm, Eu, Gd, Tb, Dy, Ho, Er, Tm, Yb, Lu).
 - Los Actínidos (Th, Pa, U, Np, Am, Cm, Br, Cf, Es, Fm, Md, No, Lw).
 - Forman la primera serie de transición los elementos del cuarto período (Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn), y la segunda serie de transición los del quinto período (Y, Zr, Nb, Mo, Tc, Ru, Rh, Cd).

- El elemento La no tiene un grupo bien definido dentro de los Metales de Transición.
- Los Halógenos son el grupo VIIb: (F, Cl, Br, I, At). El resto de elementos (C, N, O, P, S), pertenecen al resto de No-Metales.
- El grupo VIII (He, Ne, Ar, Kr, Xe, Rn) son los Gases Nobles.

4.4. Se desea construir un sistema basado en marcos que se utilizará en la enseñanza del aparato digestivo a niños de 10 años. Se sabe que el aparato digestivo lo forman los siguientes órganos: boca, esófago, estómago, intestino delgado e intestino grueso. La boca precede al esófago, y éste al estómago. A continuación, y en el siguiente orden, se encuentra el intestino delgado y el intestino grueso.

Las funciones de dichos órganos son: la boca para masticar y deglutir, el esófago para transferir alimentos; el estómago para mezclar y comenzar la digestión; el intestino delgado para absorverlos; y el intestino grueso para absorver y desecharlos.

Los órganos del aparato digestivo están unidos entre sí, pero al mismo tiempo están independizados por medios esfínteres. Así, el cardias une el esófago con el estómago e impide que el alimento pase del estómago al esófago. El piloro une el estómago y el duodeno. La válvula ileocecal une el intestino delgado con el grueso. El experto posee más conocimiento sobre estos esfínteres. Dichas propiedades se describen en otros documentos, y deberá tenerse en cuenta para incluirlas posteriormente en el sistema final.

El esófago es un tubo muscular de 30-40 cm. de longitud, cuyas contracciones empujan el bolo alimenticio desde la boca al estómago. Su mucosa no está preparada para soportar la presencia de ácido.

El estómago es un saco muscular dividido en zonas: el fundus (que está unido al esófago por el piloro) y el cuerpo (parte intermedia) para almacenar alimentos de gran tamaño; y el antro (unido al intestino delgado por el cardias) para mezclar y triturar los alimentos.

El intestino delgado está compuesto por el duodeno, yeyuno e ileón. El duodeno es un tubo de 25 cm. de longitud que conecta el estómago al resto del intestino delgado. El yeyuno sigue al duodeno, tiene dos metros de longitud, y comunica el duodeno con el ileón el cual mide 4 metros.

El intestino grueso comienza con el colon ascendente. Este precede al colon transverso, el cual va seguido del colon descendente, del colon sigmoideo, y del recto. El colon ascendente comunica el intestino delgado con el resto del intestino grueso.

En la digestión intervienen numerosos jugos, ácidos y enzimas. Además, se sabe que los jugos están compuestos por ácidos y enzimas. La enzima Amilasa está en la boca; el jugo gástrico, formado por ácido clorhídrico y la enzima Pepsina en el estómago; y, los jugos pancreáticos y biliar en el duodeno. El experto posee más conocimientos sobre ellos, y deberá tenerse en cuenta para incluir dicho conocimiento posteriormente en el sistema final.

Se pide:

- Construir el diagrama de la jerarquía de macros y explicar la semántica de las relaciones empleadas. Indicar en dicho diagrama cuáles son las propiedades de cada marco clase.
- Explicar detalladamente los marcos: intestino delgado, duodeno, yeyuno e ileón.
- Si a este sistema se le preguntaran las siguientes cuestiones, ¿cómo razonaría el sistema y qué respondería?:
 - ¿Cuáles son los componentes del estómago?
 - ¿Cuál es el esfinter superior del colon ascendente?
 - ¿Precede el estómago al esófago?
 - ¿Cuál es el órgano que está en la parte superior del cardias?
 - ¿Cuál es el órgano que está en la parte inferior del píloro?
 - ¿De qué órgano forma parte el colon transverso?
 - ¿Qué enzima produce la boca?
 - ¿Qué elementos forman el jugo pancreático?
 - ¿Es el ácido clorhídrico una enzima?
 - ¿Cuánto mide el intestino delgado?

Referencias

- FILLMORE, C. J.: «The Case for Case». En: E. Bach. y R. Harms (Eds.), *Universals in Linguistics Theory*, pp. 1–90. Holt, Rinehart and Winston Publishing Company, 1968.
- GÓMEZ, ASUNCIÓN; JURISTO, NATALIA; MONTES, CÉSAR y PAZOS, JUAN.: *Ingeniería del Conocimiento*. Editorial Centro de Estudios Ramón Areces, S.A., 1997.
- HENDRIX, G. G.: «Expanding the utility of semantic networks through partitioning». En: *Proceedings of the IJCAI-75*, pp. 115–121, 1975.
- HENDRIX, G. G.: «Encoding Knowledge in Partitioned Networks». En: N. V. Findle (Ed.), *Associative Networks - The Representation and Use of Knowledge in Computers*, pp. 51–92. Academic Press, 1979.
- MINSKY, M.: «A Framework for Representing Knowledge». En: R. J. Brachman y H. J. Levesque (Eds.), *Readings in Knowledge Representation*, pp. 245–262. Kaufmann, Los Altos, CA, 1985.
- QUILLIAN, M. R.: «Semantic Memory». En: M. Minsky (Ed.), *Semantic Information Processing*, pp. 27–70. The MIT Press, 1968.
- TOURETZKY, D.S.: *The Mathematics of Inheritance Systems*. Morgan Kaufmann Publishers, Inc, Los Altos (California), EE.UU, 1986.

Capítulo 5

Ontologías

Asunción Gómez Pérez¹, Mariano Fernández López² y Óscar Corcho¹
Universidad Politécnica de Madrid¹ y Universidad San Pablo CEU²

5.1 Introducción

Las ontologías empezaron a adquirir relevancia en el área de la informática en 1991, y más concretamente, dentro del *Knowledge Sharing Effort*, campaña emprendida por la agencia norteamericana DARPA (Agencia de Investigación de Proyectos Avanzados de Defensa) [Neches y otros, 1991]. El objetivo de esta campaña era facilitar la construcción de nuevos sistemas, basados en conocimientos, de forma diferente a la que entonces imperaba, para que las bases de conocimientos en las que estos sistemas se fundamentaban no tuvieran que construirse a partir de cero, sino ensamblando los componentes que se podían reutilizar. Esta reutilización sirve tanto para conocimientos estáticos, que se modeliza por medio de ontologías, como para el conocimiento dinámico de resolución de problemas, que se modeliza con PSMs (*Problem Solving Methods*).

Desde 1991 hasta el presente se han producido avances importantes en este área y hoy las ontologías son moneda común que se utiliza en el desarrollo de gran número de aplicaciones en áreas tan diversas como gestión del conocimiento, procesamiento de lenguaje natural, comercio electrónico, integración de información inteligente, recuperación de información, diseño e integración de bases de datos, bioinformática, educación, etc.

La llegada de la Web Semántica [Berners-Lee, 1999] ha hecho ver la necesidad, cada vez mayor, que existe de reutilizar conocimientos a la vez que ha fortalecido su propio potencial, de aquí que las ontologías y los PSMs (a los que, en muchos casos, se les puede considerar precursores de los Servicios Web Semánticos) jueguen un importante papel en este contexto.

Tal y como se describe en el título del artículo, vamos a explicar el *qué* y el *cómo* de las ontologías. En la sección 5.2 definiremos el término “ontología” y explicaremos cuáles son los principales componentes a utilizar cuando se modelizan ontologías. En las siguientes secciones vamos a mostrar cómo se desarrolla una ontología siguiendo METHONTOLOGY y utilizando WebODE (metodología y plataforma propuestas

por el Grupo de Ingeniería Ontológica de la Universidad Politécnica de Madrid). Con esta tecnología nuestro grupo ha construido ontologías para campos tan dispares como el de la química, las ciencias naturales, la gestión del conocimiento, el comercio electrónico, etcétera. El ejemplo que mostramos en este artículo es la adaptación de una taxonomía de clase propuesta por Breuker¹, para construir una ontología de personas jurídicas en el contexto del sistema legal español, para la cual utilizamos METHONTOLOGY (véase la sección 5.3) y WebODE (véase la sección 5.4) como soporte tecnológico. La sección 5.5 describe de forma somera otros métodos y metodologías, así como algunas de las herramientas existentes.

En la sección 5.6 mostraremos brevemente una panorámica de los lenguajes que existen para implementar ontologías, y por último, en la sección 5.7 expondremos las conclusiones. Al final de estas secciones, ofreceremos una lista de enlaces a ontologías y una serie de ejercicios (tanto resueltos como no resueltos)².

5.2 Ontologías: definición y componentes

Desde la Antigua Grecia hasta nuestros días se ha estudiado *ontología*, aquella rama del saber que trata sobre la esencia de las cosas que permanece a través de los cambios. Uno de los frutos de la ontología ha sido las ontologías. Para un filósofo, *una ontología* es “*un sistema particular de categorías sistematizando cierta visión del mundo*” [Guarino, 1998]. Un ejemplo de ontología es la de [Lowe, 2006], que distingue entre entes universales (p. ej. ordenador) y particulares (p. ej. este ordenador), y entre sustancias (p. ej. este ordenador) y modos (p. ej. el color de este ordenador).

Los herederos en informática de las ontologías filosóficas son aquellos artefactos compatibles y reutilizables que tienen que ser desarrollados en un lenguaje comprensible para el ordenador [Gruber, 2006; Studer y otros, 1998]. Esta idea está expresada en la definición aportada por [Studer y otros, 1998]: *una ontología es una especificación formal de una conceptualización compartida*. Para nosotros, esta definición es, sin duda, la más completa de las encontradas en la literatura.

Existen diferentes formalismos de representación de conocimientos para formalizar (e implementar) ontologías y cada uno de ellos tiene distintos componentes que pueden ser utilizados en estas tareas de formalización e implementación, sin embargo, dichos formalismos comparten un conjunto mínimo de componentes, a saber:

- **Clases**, que representan conceptos tomados en su sentido más amplio. En el dominio del los viajes, por ejemplo, los conceptos que tenemos son: lugares (ciudades, pueblos, etc.), alojamiento (hoteles, campings, etc.) y medios de transporte (aviones, trenes, coches, transbordadores, motos y barcos). En la ontología, las clases están normalmente organizadas en taxonomías por medio de las cuales se pueden aplicar los mecanismos de herencia. Podemos representar, por ejemplo, una taxonomía de lugares de diversión (teatro, cine, sala de conciertos, etc.)

¹<http://zeus.ics.forth.gr/forth/ics/isl/projects/ontoweb/notes/legal-ontol-ontoweb-sard-2002.ppt>

²Para profundizar en el campo de la ingeniería ontológica recomendamos el libro Gómez-Pérez y otros [Gómez-Pérez y otros, 2003].

o de viajes organizados (billete clase turista, billete clase preferente, etc.) Por otro lado, las metaclasses también se pueden definir en el paradigma KR basado en marcos. Metaclasses son clases cuyas instancias son también clases y permiten hacer una gradación del significado ya que establecen diferentes capas de clases en la ontología en la que están definidas.

- **Relaciones**, que representan un tipo de asociación entre los conceptos del dominio y se definen formalmente como cualquier subconjunto de un producto de n conjuntos, es decir: $R \subset C_1 x C_2 x \dots x C_n$. Las ontologías normalmente contienen relaciones binarias, cuyo primer argumento es conocido como dominio de la relación, mientras que el segundo es conocido como rango. Por ejemplo, la relación binaria **lugar de llegada** tiene el concepto **viaje** como dominio y el concepto **lugar** como rango. Las relaciones pueden instanciarse con conocimientos del dominio, por ejemplo, para expresar que el vuelo AA7462-Feb-08-2002 llega a Seattle tenemos que escribir: (**lugarDeLlegada AA7462-Feb-08-2002 Seattle**).

Las relaciones binarias se utilizan algunas veces para expresar atributos de conceptos, conocidos como ranuras (*slots*) y estos atributos se distinguen de las relaciones porque su rango es un tipo de datos como, por ejemplo, cadena de caracteres, número, etc., mientras que el rango de las relaciones es un concepto.

- **Axiomas**. Según [Gruber, 1992] sirven para modelizar afirmaciones que son siempre ciertas. En ontologías se utilizan generalmente para representar conocimiento que no se puede definir formalmente a través de otros componentes; además sirven para verificar la consistencia de la ontología misma o la consistencia de los conocimientos almacenados en una base de conocimientos, por lo que son muy útiles para inferir conocimientos nuevos. Un axioma en el dominio de los viajes sería: *no es posible viajar de América a Europa en tren*.
- **Instancias**, que se utilizan para representar elementos o individuos en una ontología. Un ejemplo de instancia del concepto AA7462 es el vuelo AA7462 que llega a Seattle el 8 de febrero del 2006 y que cuesta 300 (dólares americanos, euros o cualquier otra moneda).

Las ontologías se pueden formalizar, además de con formalismos y lenguajes creados específicamente para representar conocimientos a través de enfoques del campo de la ingeniería del software, tales como el *Unified Modeling Language* (UML) [Rumbaugh y otros, 1998] o los diagramas entidad relación (ER) [Chen, 1976].

En este contexto, el equipo del *Object Management Group* (OMG) está trabajando en una especificación que pueda definir los metamodelos de algunos de los tipos de diagramas y lenguajes que se utilizan en la representación de ontologías. Esta especificación, que es conocida con el nombre *Ontology Description Model* (ODM), utiliza para describir metamodelos una notación formal. Los metamodelos (que ya han sido definidos tanto para UML como para el entidad relación, OWL, RDF(S) etc.) pueden considerarse formalizaciones de ontologías de representación del conocimiento.

Las correspondencias entre metamodelos se han descrito formalmente en el documento de ODM [OMG, 2005].

El objetivo de dicho documento es hacer que los ingenieros de software puedan modelar ontologías con notaciones que les sean conocidas como, por ejemplo, UML y ER, y transformar sus modelos conceptuales en ontologías formales que se puedan representar en lenguajes de ontologías.

5.3 Una metodología para el desarrollo de ontologías: METHONTOLOGY

METHONTOLOGY [Corcho y otros, 2007; Fernández-López y otros, 1997, 1999] es una metodología diseñada por el Grupo de Ingeniería Ontológica de la Universidad Politécnica de Madrid (UPM) que permite construir ontologías en el nivel de conocimientos, y que se elaboró partiendo de las principales actividades identificadas en el proceso de desarrollo de software de IEEE [IEEE, 1996], y de otras metodologías de ingeniería del conocimiento [Gómez-Pérez y otros, 1997].

Las herramientas ODE y WebODE [Arpírez y otros, 2003] fueron creadas para dar apoyo tecnológico a METHONTOLOGY; sin embargo, existen otras que también se pueden utilizar para construir ontologías siguiendo este método como son: Protégé-2000 [Noy y otros, 2000], OntoEdit [Benton y Kambhampati, 2002], KAON [Maedche y otros, 2003], etcétera. Nuestra metodología, METHONTOLOGY, ha sido recomendada por la *Foundation for Intelligent Physical Agents* (FIPA)³ para construir ontologías. Esta organización aboga por la interoperabilidad de las aplicaciones basadas en agentes. METHONTOLOGY propone un proceso de desarrollo, un ciclo de vida de la ontología y una especificación para cada una de las actividades que se realizan durante el proceso de desarrollo.

En las siguientes secciones vamos a describir qué entendemos por proceso de desarrollo de ontologías y qué por su ciclo de vida. Después vamos a mostrar el proceso que hemos seguido para conceptualizar una ontología de personas jurídicas (personas jurídicas, organizaciones, etc.) en el dominio del sistema legal español. Como ya hemos comentado previamente, hemos adaptado la taxonomía de clases de personas jurídicas propuesta por Breuker⁴ al sistema legal español. Las definiciones que ofrecemos para algunos de los términos legales de nuestra ontología son adaptaciones del lexicón de LEGAMedia⁵.

5.3.1 Proceso de desarrollo de ontologías y su ciclo de vida

El proceso de desarrollo de ontologías y su ciclo vida ya fue dado a conocer por [Fernández-López y otros, 1997] en el marco de METHONTOLOGY, y sus propuestas estaban basadas en el estándar que IEEE establece para el desarrollo de software [IEEE, 1996]. Dicho proceso supone llevar a cabo las actividades necesarias para

³<http://www.fipa.org/specs/fipa00086/>

⁴<http://zeus.ics.forth.gr/forth/ics/sl/projects/ontoweb/notes/legal-ontol-ontoweb-sard-2002.ppt>

⁵<http://www.legamedia.net/lx/lx.php>

construir ontologías y que pueden clasificarse en tres categorías (véase la Figura 5.1): Actividades de gestión, desarrollo y apoyo.

Las **actividades de gestión** incluyen una *planificación*, así como un *control* y una *garantía de calidad*. La actividad del planificación indica las tareas que hay que realizar, y en qué orden, así como el tiempo y los recursos que se necesitan para completarlas. Esta actividad es esencial para ontologías que utilizan a su vez ontologías almacenadas en bibliotecas de ontologías, o para las que requieren una gran cantidad de abstracción y de generalidad. La actividad de *control* garantiza que las tareas programadas se terminen de la forma deseada y la actividad de garantía de calidad nos asegura que la calidad de todos y cada uno de los productos construidos (ontología, programas y documentación) es satisfactoria.

Las **actividades orientadas al desarrollo**, tal y como aparecen en la Figura 5.1, se pueden clasificar en tres grupos: actividades previas al desarrollo, desarrollo propiamente dicho y actividades posteriores. Durante las actividades previas al desarrollo, un *estudio del entorno* identifica el problema a resolver con la ontología, las aplicaciones en donde la ontología tiene que integrarse, etc. y también se lleva a cabo un *estudio de viabilidad* que responde a preguntas tales como: ¿es posible construir una ontología?, ¿es conveniente construirla?, etc.

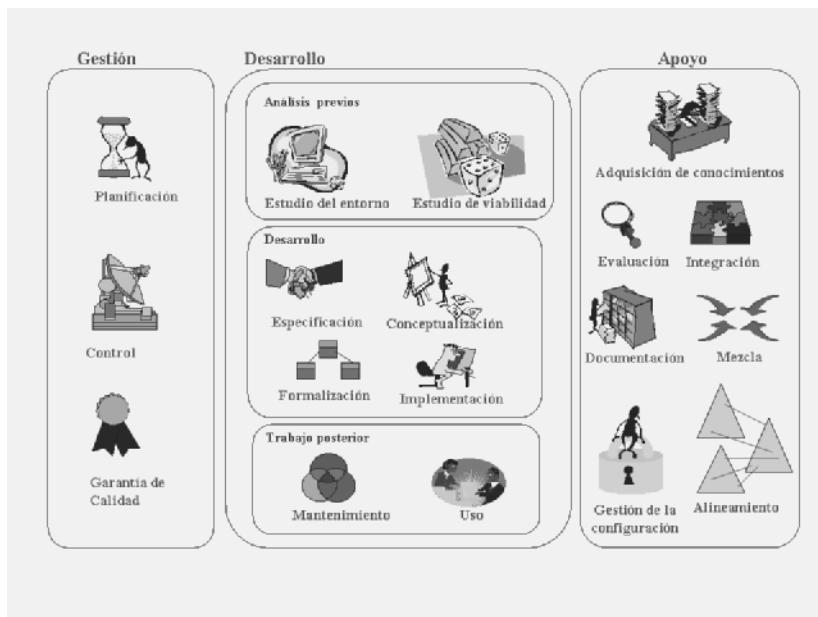


Figura 5.1: Proceso de desarrollo de ontologías (adaptado de [Fernández-López y otros, 1997]).

Durante las actividades de desarrollo propiamente dicho, la tarea de *especificación* establece la razón por la cual se construye la ontología, qué usos se quiere que tenga y quiénes son los usuarios finales, mientras que la tarea de *conceptualización* estructura

el conocimiento del dominio como modelos con significado bien partiendo de cero, bien reutilizando modelos ya existentes; en este último caso, se realizan actividades tales como recortar ramas de las taxonomías existentes, ampliar la cobertura de las ontologías añadiéndoles nuevos conceptos a los niveles más altos de sus taxonomías, o especializar las ramas que requieren mayor granularidad. Dicha actividad es independiente del lenguaje de implementación. Por otro lado, la tarea de formalización consiste en transformar el modelo conceptual en un modelo formal o semicomputable; por último, la *actividad de implementación* crea modelos computables en un lenguaje de ontologías.

En el periodo posterior al desarrollo, la actividad de *mantenimiento* consiste en actualizar y corregir la ontología si fuera necesario; en este periodo la ontología puede reutilizarse por otras ontologías o aplicaciones. La actividad de *evolución* consiste en controlar los cambios que se producen en la ontología y sus efectos creando y ocupándose de las diferentes variantes de la ontología y teniendo en cuenta que dichas variantes pueden utilizarse en distintas ontologías y aplicaciones. [Noy, 2006].

Por ultimo, las **actividades de apoyo** incluyen una serie de tareas que se pueden realizar durante las actividades dedicadas al desarrollo y sin las cuales la ontología no se podría construir. En estas tareas se incluyen la de *adquisición de conocimientos, evaluación, integración, mezcla, alineamiento, documentación y gestión o control de la configuración*. El objetivo de la tarea de *adquisición de conocimientos* es adquirir los conocimientos que los distintos expertos tienen sobre un determinados dominio o adquirir conocimientos a través de algún tipo de proceso semiautomático y que llamamos “aprendizaje de ontologías” [Kietz y otros, 2000]. La actividad de *evaluación* [Gómez-Pérez, 1994] consiste en emitir un juicio técnico sobre las ontologías, sobre sus entornos de software asociados y sobre su documentación. Dicho juicio se hace respecto a un marco de referencia, tanto durante cada fase como entre las fases del ciclo de vida de la ontología. Por otro lado, la actividad de *integración* es necesaria cuando se construye una ontología reutilizando otras ontologías disponibles. Otra actividad de apoyo que hay que mencionar es la de *mezcla* [Gangemi y otros, 1999; Noy y Musen, 2000; Steve y otros, 1998; Stumme y Maedche, 2001], que consiste en extraer una nueva ontología de otras ontologías del mismo dominio. La ontología así obtenida puede unificar conceptos, terminología, definiciones, restricciones, etc, de las ontologías fuente. La mezcla de dos o más ontologías se puede realizar bien durante el tiempo de ejecución, bien durante el diseño. Durante la actividad de *alineamiento* se establecen diferentes tipos de correspondencias entre las ontologías implicadas, de tal forma que las ontologías originales no se fusionan. La actividad de *documentación* detalla de forma clara y exhaustiva cada uno de los estadios completados, así como los productos creados. En cuanto a la actividad de *gestión de la configuración*, diremos que consiste en registrar todas las versiones de la documentación y del código de la ontología con objeto de controlar los cambios. También se puede tener en cuenta una *actividad multilingüe* que consiste en establecer correspondencias entre ontologías y descripciones formales de conocimientos lingüísticos [Declerck y Uszkoreit, 2003]; esta actividad no es siempre considerada una actividad de apoyo, sin embargo, es muy pertinente en el contexto de las ontologías en red que están disponibles en la Web Semántica.

El proceso de desarrollo de ontologías no explica en qué orden hay que realizar las actividades ya que éste es el cometido del **ciclo de vida de la ontología**, el cual se encarga de expresar *cuándo* se tienen que hacer, es decir establece el conjunto de etapas o fases que atraviesa la ontología durante su vida útil, y describe las actividades que hay que realizar en cada fase y la relación entre las fases (relación de precedencia, regreso o retorno, etc.).

La primera versión del modelo del proceso del ciclo vida de METHONTOLOGY (véase la Figura 5.2) propone comenzar con una programación de las actividades a realizar. Y después sigue con la actividad de *especificación*, que consiste en explicar por qué hay que construir la ontología, qué posibles usos tendrá, y quiénes serán sus usuarios. Cuando la tarea de especificación termina, comienza la de conceptualización, cuyo objetivo es organizar y estructurar los conocimientos adquiridos, utilizando para ello un conjunto de representaciones que los expertos del dominio pueden manipular fácilmente. Despues de que se ha construido el modelo conceptual éste tiene que ser formalizado e implementado (pero si el modelo conceptual es lo suficientemente formal no será necesario pasar por estas dos fases, sino que se puede ir directamente a la implementación). Para ver más detalles, consúltese [Gómez-Pérez y otros, 2003].

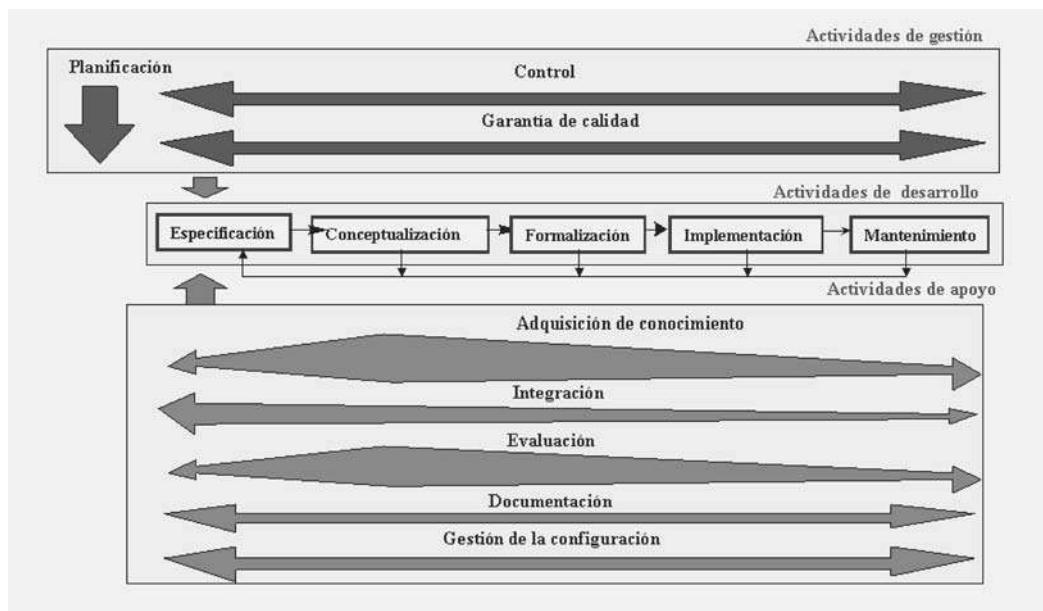


Figura 5.2: Ciclo de vida de la ontología en METHONTOLOGY.

El ciclo de vida de la Figura 5.2 ha sido modificado, ya que existe un gran número de ontologías disponibles tanto en bibliotecas de ontologías como en Internet, por lo que su reutilización por otras ontologías y aplicaciones ha aumentado. Las ontologías de dominio pueden ser reutilizadas para construir otras de mayor granularidad y cobertura y además se pueden mezclar con otras para crear unas nuevas. Haciendo

una analogía con el mapa de un metro subterráneo, podemos observar que existe una línea principal (en el centro de la Figura 5.3) que propone las principales actividades de desarrollo ya señaladas en las anteriores versiones de METHONTOLOGY. Hay otras líneas que nacen de la principal o terminan allí y otras van en paralelo y se bifurcan en un punto. Es por esto que surgen relaciones de interdependencia [Gómez-Pérez y Rojas, 1999] entre el ciclo de vida de varias ontologías, por lo que las acciones de evaluación, recorte y mezcla se pueden realizar en dichas ontologías, es decir, los ciclos de vida de las diferentes ontologías se cruzan, lo que produce diferentes escenarios con diferentes requisitos tecnológicos. [Corcho y otros, 2007] han descrito algunos de los escenarios más comunes que suelen aparecer en este contexto.

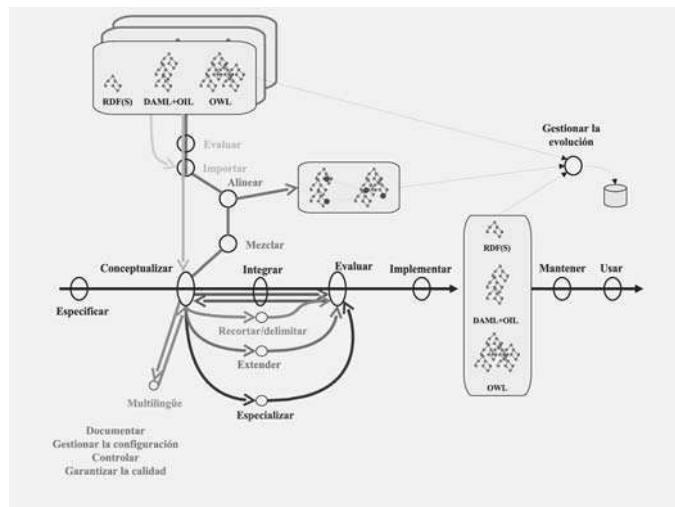


Figura 5.3: Proceso de desarrollo de ontologías en red.

5.3.2 Conceptualización de una ontología de entidades legales

Tanto el proceso para desarrollar ontologías como sus ciclos de vida han sido ya identificados por [Fernández-López y otros, 1997] en el marco de METHONTOLOGY. Sus propuestas se basan en el estándar del IEEE para el desarrollo de software [IEEE, 1996].

Al construir ontologías, durante la fase conceptualización el ingeniero no debe ser anárquico en el uso de los componentes de modelización arriba señalados, ni debe definir, por ejemplo, una relación si los conceptos conectados no están definidos en la ontología de forma precisa. METHONTOLOGY incluye en esta fase el conjunto de tareas de conocimiento estructurado que aparece en la Figura 5.4. En la figura se destacan los componentes de la ontología (conceptos, atributos, relaciones, constantes, axiomas formales, reglas e instancias) construidos dentro de cada tarea y muestra el orden que se ha propuesto para crear tales componentes durante la actividad de

conceptualización. Este proceso de modelización no es secuencial aunque haya que seguir un cierto orden para asegurar la consistencia y completitud del conocimiento representado. Si se introduce nuevo vocabulario el ingeniero puede volver a la tarea anterior.

Tarea 1: Construir el glosario de términos. Se debe primeramente construir un glosario de términos que incluya todos los términos relevantes del dominio (conceptos, instancias, atributos, relaciones entre conceptos, etc.), las descripciones de los términos para el lenguaje natural, así como sus antónimos y sinónimos. La Tabla 5.1 muestra una sección del glosario de términos de la ontología de entidades legales. Es importante comentar aquí que en las primeras fases de la conceptualización, el glosario de términos puede contener varios términos que hacen referencia al mismo componente; en ese caso, hay que hacerlos figurar como sinónimos.

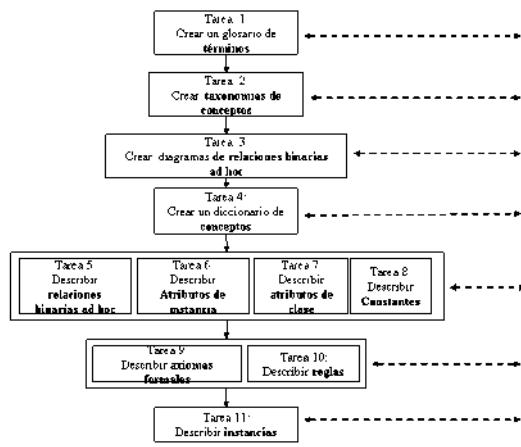


Figura 5.4: Tareas de la actividad de conceptualización según METHONTOLOGY.

Término	Sinónimos	Acrónimos	Descripción	Tipo
Mayoría de edad en España	— —	— —	La mayoría de edad en España es de 18 años	Constant
Tribunal	Tribunal de justicia	— — — —	Aunque tribunal puede ser entendido como un lugar físico, asumimos en esta ontología que se trata un conjunto de personas.	Concept
Fecha de nacimiento	— —	— —	El día que nació una persona.	Instance Attribute
Abogado defensor	— —	— —	Quien lleva la defensa en un pleito.	Relation

Tabla 5.1: Fragmento del glosario de términos de la ontología de entidades legales.

Tarea 2: Construir taxonomías de conceptos. Cuando el glosario contiene un número de términos considerable, hay que construir taxonomías de conceptos para

establecer una jerarquía; para ello, se seleccionan del glosario de términos los términos que son conceptos. METHONTOLOGY propone utilizar las cuatro relaciones taxonómicas definidas en la *Frame Ontology* [Farquhar y otros, 1997] y en la Ontología OKBC, a saber: subclase de, descomposición disjunta y descomposición exhaustiva.

Un concepto C_1 es una subclase de otro concepto C_2 si y sólo si cada instancia de C_1 es también una instancia de C_2 . En la Figura 5.5 podemos ver cómo **persona física** es una subclase de **persona**, ya que cada persona física es una persona. Un concepto puede ser una subclase de más de un concepto de la taxonomía, por ejemplo, el concepto **empresa de control compartido** es una subclase de los conceptos **empresa privada** y **empresa pública**, puesto que una compañía cuya dirección está compartida puede estar dirigida por entidades tanto públicas como privadas.

Una *descomposición disjunta* de un concepto C es un conjunto de subclases de C que no tienen instancias comunes y que no tienen por qué cubrir C, es decir, puede haber instancias del concepto C que no son instancias de ningún concepto en la descomposición. Por ejemplo (véase la Figura 5.5) los conceptos **ministerio** y **tribunal** forman una descomposición disjunta del concepto **organización** ya que ninguna organización puede ser a la vez un ministerio y un tribunal. Además, puede haber instancias del concepto **organización** que no son instancias de ninguna de las dos clases.

Una *descomposición exhaustiva* de un concepto C es un conjunto de subclases de C que cubren C, y que pueden tener instancias y subclases comunes, es decir, no puede haber instancias del concepto C que no sean instancias de por lo menos uno de los conceptos de la descomposición. Por ejemplo (véase la Figura 5.5) los conceptos **empresa privada** y **empresa pública** forman una descomposición exhaustiva del concepto **empresa** porque no hay compañías que no sean instancias de por lo menos uno de dichos conceptos, aunque pueden tener instancias comunes. Por ejemplo, una **empresa de control compartido** es una **empresa pública** y una **empresa privada**.

Una *partición* de un concepto C es un conjunto de subclases de C que no tienen ninguna instancia en común y que cubre C, es decir, no hay instancias de C que no sean instancias de uno de los conceptos en la partición. Por ejemplo, la Figura 5.5 muestra que los conceptos **menor de edad** y **mayor de edad** forman una partición del concepto **persona física**, porque cada persona física es o menor o mayor de edad. Después de haber estructurado la taxonomía de conceptos y antes de seguir con la especificación de nuevos conocimientos, se deberá comprobar que las taxonomías no contienen errores [Gómez-Pérez, 2006]. Por ejemplo, se deberá comprobar que un elemento no es simultáneamente una instancia de dos clases de una descomposición disjunta, que no hay bucles en la taxonomía de conceptos, que varios términos no tienen el mismo significado, etc.

Tarea 3: Construir diagramas de relaciones binarias *ad hoc*. Una vez que se ha construido y evaluado la taxonomía, hay que construir diagramas de relaciones binarias *ad hoc* tal y como propone la actividad de conceptualización. El objetivo de este diagrama es establecer relaciones *ad hoc* entre conceptos de la misma (o diferente) taxonomía de conceptos. En la Figura 5.6 podemos ver un fragmento del diagrama de relaciones binarias *ad hoc* de nuestra ontología de entidades legales con la relación **es demandante**, **es demandado** y **ve**, así como sus contrarios **tiene**

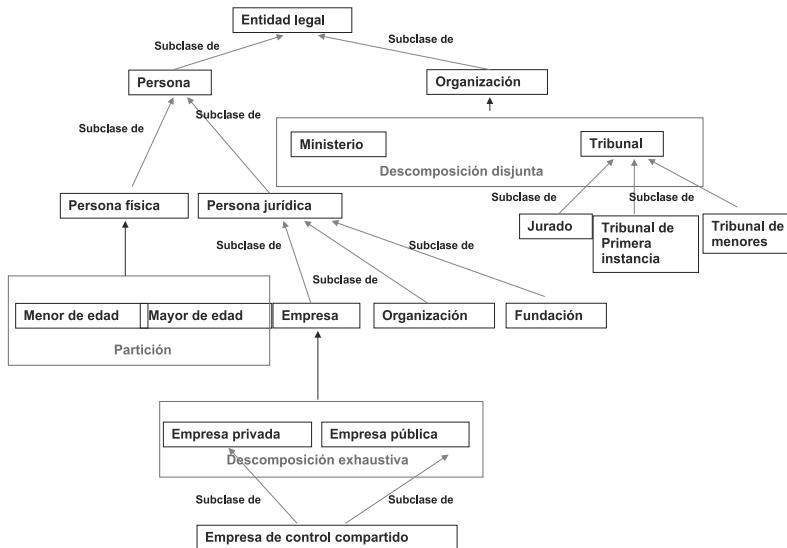


Figura 5.5: Una parte de la taxonomía de conceptos de una ontología de entidades legales.

demandante, tiene demandado y es visto. Tales relaciones conectan los conceptos origen (persona y pleito; tribunal y pleito) de las taxonomías de conceptos de entidades jurídicas y pleitos. Desde la perspectiva de la integración de la ontología, dichas relaciones *ad hoc* significan que la ontología de entidades legales incluirá la ontología de pleitos y viceversa.

Antes de seguir adelante con la especificación de nuevos conocimientos, se deberá comprobar que el diagrama de relaciones binarias *ad hoc* no contenga ningún error y descubrir si los dominios y rangos de cada argumento de cada relación delimitan de forma precisa y exacta las clases que son apropiadas para la relación. Hay que tener en cuenta que los errores aparecen cuando los dominios y los rangos son imprecisos o excesivamente concretos.

Tarea 4: Construir el diccionario de conceptos. Una vez que se han creado las taxonomías de conceptos y los diagramas de relaciones binarias *ad hoc*, se debe especificar cuáles son las propiedades y las relaciones que describen cada concepto de la taxonomía en un diccionario de conceptos y, opcionalmente, también pueden aparecer sus instancias.

Un diccionario de conceptos contiene todos los conceptos del dominio, sus relaciones, sus instancias, así como sus atributos de clase e instancia. Las relaciones que se especifican para cada concepto son las que tienen por dominio el concepto. Por ejemplo, el concepto persona tiene dos relaciones: es demandante y es demandado. Las relaciones y los atributos de instancia y de clase pertenecen a los conceptos, lo que significa que sus nombres se pueden repetir en diferentes conceptos. La Tabla 5.2

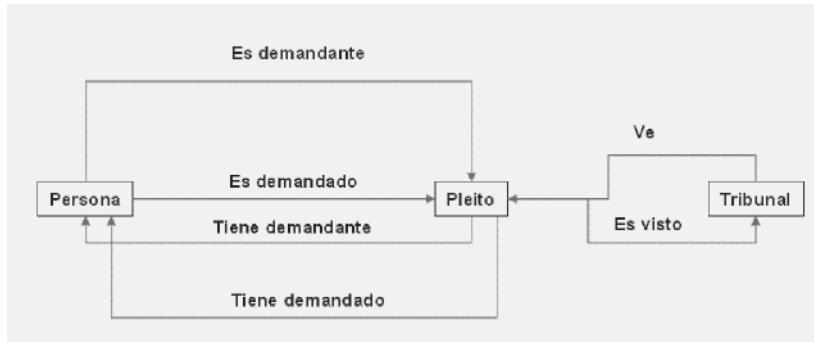


Figura 5.6: Parte del diagrama de relaciones binarias *ad hoc* de la ontología de entidades legales.

muestra una pequeña sección del diccionario de conceptos de la ontología de entidades legales.

Como ya hemos comentado, después de construir el diccionario, se debe describir minuciosamente cada una de las relaciones binarias *ad hoc*, los atributos de clase y los atributos de instancia que aparecen en él y, además, hay que describir de forma pormenorizada cada una de las constantes que aparecen en el glosario de términos. Si bien METHONTOLOGY establece cómo realizar estas tareas, no propone, sin embargo, el orden en el que deben hacerse.

Tarea 5: Definir minuciosamente las relaciones binarias *ad hoc*. El objetivo de esta tarea es describir todas las relaciones binarias *ad hoc* incluidas en el diccionario de conceptos y producir la tabla de relaciones binarias *ad hoc*. Para cada relación binaria *ad hoc*, hay que especificar su nombre, así como los nombres de los conceptos origen y destino, su cardinalidad y su relación inversa. La Tabla 5.3 muestra una sección de la relación binaria *ad hoc* de la ontología de las entidades legales que contiene la definición de las relaciones *es demandado*, *es demandante*, etc.

Tarea 6: Definir detalladamente los atributos de instancia. El objetivo de esta tarea es describir todos los atributos de instancia ya incluidos en el diccionario de conceptos. Cada fila de la tabla de atributos de instancia contiene una descripción pormenorizada de un atributo de instancia. Los atributos de instancia describen las instancias del concepto y su(s) valor(es) pueden ser diferentes para cada instancia del concepto. Para cada atributo de instancia tenemos que especificar los siguientes campos: el nombre, el concepto al que pertenece (los atributos están en o pertenecen a los conceptos), el tipo de valor, el intervalo de valores (en el caso de valores numéricos), la cardinalidad mínima y máxima, los atributos de instancia, los atributos de clase y constantes que se han utilizado para inferir los valores del atributo, los atributos que se pueden deducir con valores de este atributo, las fórmulas o reglas que permitan inferir valores del atributo y las referencias utilizadas para definir el atributo. La Tabla 5.4 muestra un fragmento de los atributos de instancia de la ontología de entidades legales. Algunos de los campos mencionados anteriormente no aparecen por cuestiones

<i>Concepto</i>	<i>Instancias</i>	<i>Atributos de Clase</i>	<i>Atributos de Instancia</i>	<i>Relaciones</i>
Tribunal	Tribunal de la Audiencia Nacional Tribunal Constitucional Tribunal Supremo Tribunal Provincial de Albacete	–	Número de miembros Sede Jurisdicción territorial	Ve
Empresa	–	Tipo de control	Nombre	–
Abogado defensor	–	–	–	–
Persona	–	–	–	Es demandado Es demandante
Persona física	–	–	Edad Fecha de nacimiento Fecha de muerte Nombre Apellidos Nacionalidad	–

Tabla 5.2: Parte del diccionario de conceptos de la ontología de entidades legales.

Relación	Concepto Origen	Cardinalidad (Max)	Concepto Destino	Relación Inversa
Es demandado	Person	N	pleito	Tiene demandado
Es demandante	Person	N	pleito	Tiene demandante
Ve	Court	N	pleito	es visto

Tabla 5.3: Parte de la tabla de relaciones binarias *ad hoc* de la ontología de entidades legales.

de espacio. Dicha tabla contiene algunos de los atributos de instancia del concepto **tribunal**: **número de miembros**, **sede** y **jurisdicción territorial**.

El haber utilizado unidades de medida para los atributos numéricos da como resultado la integración de la ontología *Standard Units*. Esto es un ejemplo de cómo METHONTOLOGY propone integrar ontologías durante la actividad de conceptualización, en vez de posponer dicha integración a la actividad de implementación.

Atributo de Instancia	Concepto	Tipo	Dominio Valor	Cardinalidad
Número de miembros	Tribunal	Integer	1 ..	(1, 1)
Sede	Tribunal	String	—	(1, 1)
Jurisdicción territorial	Tribunal	String	—	(1, 1)

Tabla 5.4: Parte de la tabla de atributos de instancia de la ontología de entidades legales.

Tarea 7: Definir minuciosamente los atributos de clase. El objetivo de esta tarea es describir los atributos de clase que ya están incluidos en el diccionario de conceptos usando una tabla de atributos de clase. Cada fila de la tabla de los atributos de clase contiene su descripción detallada. Para cada atributo de clase hay que incluir la siguiente información: el nombre, el nombre del concepto en el que se define el atributo, tipo de valor, valor(es), cardinalidad, los atributos de instancia cuyos valores pueden inferirse con el valor de este atributo de clase, etc. Por ejemplo, el atributo de clase, **tipo de control**, estaría definido por los conceptos de **empresa privada** y **empresa pública**, tal y como aparecen en la Tabla 5.5.

Atributo de Clase	Concepto	Tipo	Cardinalidad	Valores
Tipo de control	Empresa privada	[privado,público]	(1,2)	Privado
Tipo de control	Empresa pública	[privado,público]	(1,2)	Público

Tabla 5.5: Parte de la tabla de atributos de clase de la ontología de entidades legales.

Tarea 8: Definir las constantes minuciosamente. El objetivo de esta tarea es describir cada una de las constantes definidas en el glosario de términos. Cada fila de la tabla de constantes contiene una detallada descripción de una constante, y para cada constante hay que especificar el nombre, el tipo de valor (un número, una masa, etc.), el valor, la unidad de medida para las constantes numéricas, así como los atributos que se pueden inferir con la constante. La Tabla 5.6 muestra un fragmento de la tabla de constantes de nuestra ontología de entidades jurídicas, en donde se define

la constante **mayoría de edad en España** y en donde se han omitido los atributos que se pueden inferir con la constante.

<i>Constante</i>	<i>Tipo</i>	<i>Valor</i>	<i>Unidad de Medida</i>
Mayoría de edad en España	Cardinal	18	año

Tabla 5.6: Parte de la tabla de constantes de la ontología de entidades legales.

METHONTOLOGY propone describir axiomas formales y reglas en paralelo después de que se hayan definido los conceptos y sus taxonomías, las relaciones **ad hoc**, los atributos y las constantes.

Tarea 9: Definir axiomas formales. Para realizar esta tarea primero hay que identificar los axiomas formales que la ontología necesita y después describirlos detalladamente. Para cada definición formal de un axioma, METHONTOLOGY propone determinar claramente la siguiente información: el nombre, la descripción en lenguaje natural, la expresión que describe formalmente el axioma con lógica de primer orden, los conceptos, los atributos y las relaciones *ad hoc* a las que el axioma se refiere y las variables que se han utilizado.

Como ya se ha comentado, METHONTOLOGY propone establecer axiomas formales en lógica de primer orden. La Tabla 5.7 muestra un axioma formal de nuestra ontología de entidades legales que estipula que una persona no puede ser demandante y demandado en el mismo pleito. Las columnas que corresponden a los conceptos ya referidos y a las relaciones contienen los conceptos y las relaciones que se utilizan en el axioma formal. Las variables utilizadas son ?X para **persona** e ?Y para **pleito**.

Hemos de comentar que la definición de la expresión lógica puede resultar difícil para un experto que no tenga experiencia en lógica de primer orden.

Tarea 10: Definir las reglas. Al igual que en la tarea anterior, hay que identificar primero qué reglas se necesitan en la ontología, y luego describirlas en la tabla de reglas. Para definir cada regla, METHONTOLOGY propone incluir la siguiente información: nombre, descripción en lenguaje natural, la expresión que describe formalmente la regla, los conceptos, atributos y relaciones a los que la regla se refiere, y las variables que se utilizan en la expresión.

Además, METHONTOLOGY propone especificar las expresiones de reglas por medio de la plantilla *if <condiciones>then <consecuente>*. El lado izquierdo de la regla está formado por conjunciones de átomos mientras que el lado derecho está formado por un solo átomo.

La Tabla 5.8 muestra una regla que estipula que los juicios en los que los demandados tienen 14 años o menos, deben celebrarse en un tribunal de menores. Esta regla nos permite inferir el tipo de tribunal. Como aparece en la tabla, la regla hace referencia a los conceptos **menor de edad**, **pleito** y **tribunal**, al atributo **edad**, y a las relaciones **es demandado** y **ve**, y las variables empleadas son ?X para **menor de edad**, ?Y para **integer**, **pleito** para ?Z y ?Z para **tribunal**.

Al igual que en el caso de los axiomas formales, definir la expresión de la regla puede resultar difícil para los expertos con poca experiencia en lógica de primer orden.

Axioma	Descripción	Expresión	Conceptos Asociados	Relaciones Asociadas	Variables
Incompatibilidad demandante-demandado	Una persona no puede ser demandante y demandado en el mismo pleito	not (exists (?X,?Y) (persona (?X) and pleito (?Y) and [es demandante] (?X,?Y) and [es demandado] (?X,?Y))))	persona pleito	Es demandante, Es demandado	?X ?Y

Tabla 5.7: Parte de la tabla de axiomas de la ontología de entidades legales.

Regla	Descripción	Expresión	Conceptos	Atributos Afectados	Relaciones Afectadas	Variables
Tribunal de menores para menores	Los juicios en los que los demandados tienen 14 años o menos, deben celebrarse en un tribunal de menores	If [menor de edad](?X) and pleito(?Z) and tribunal(?W) and edad(?X, ?Y) and ?Y >14 and [es demandado](?X, ?Z) and ve(?W, ?Z) then [tribunal de menores](?W)]	Menor de edad Pleito Tribunal	Edad	Es demandado Ve	?X ?Z ?W

Tabla 5.8: Parte de la tabla de reglas de la ontología de entidades legales.

Tarea 11: Definir instancias. Después de crear el modelo conceptual de la ontología, hay que definir instancias relevantes que aparecerán en el diccionario de conceptos y dentro de una tabla de instancias. Para cada instancia hay que definir lo siguiente: su nombre, el nombre del concepto al que pertenece, y sus valores en los atributos, si se conocen. La Tabla 5.9 muestra algunas instancias de la tabla de instancias de nuestra ontología de entidades legales: **Audiencia Nacional, Tribunal Supremo y Tribunal Constitucional**, que son todas ellas instancias del concepto **tribunal**, tal y como aparece definido en el diccionario de conceptos, y tienen algunos valores de atributo y relación especificadas para : **sede, jurisdicción territorial, y número de miembros**. Dichas instancias pueden tener más de un valor para los atributos cuya cardinalidad máxima es superior a uno.

Instancia	Concepto	Attributo	Valores
Tribunal de la Audiencia Nacional	Tribunal	Sede	Madrid
		Jurisdicción territorial	España
Tribunal Supremo	Tribunal	Jurisdicción territorial	España
		Número de miembros	12
Tribunal Constitucional	Tribunal	Jurisdicción territorial	España

Tabla 5.9: Parte de la tabla de instancias de la ontología de entidades legales.

Diferentes grupos han utilizado METHONTOLOGY para construir ontologías de química, ciencias naturales, gestión de conocimientos, comercio electrónico, etc. Para una descripción más detallada de esta metodología para construir ontologías ver [Gómez-Pérez y otros, 2003].

5.4 Cómo construir una ontología legal con WebODE

WebODE⁶ [Arpírez y otros, 2003] es una plataforma de ingeniería ontológica desarrollada por el Grupo de Ingeniería Ontológica de la Universidad Politécnica de Madrid (UPM) cuya versión actual es la 2.0. WebODE es sucesor directo de ODE (entorno para diseñar ontologías), una herramienta para crear ontologías que se basa en tablas y grafos, y permite a los usuarios personalizar el modelo de conocimientos utilizado para conceptualizar las ontologías según las necesidades que tengan de representación de conocimientos. Tanto ODE como WebODE sirven de apoyo a METHONTOLOGY, la metodología para construir ontologías, que ya hemos descrito en la sección anterior.

La versión actual de WebODE contiene un editor de ontologías, que incluye casi todos los servicios que la plataforma ofrece, además de un sistema de gestión de conocimientos basado en ontologías (ODEKM), un generador automático de portales de la Web Semántica (ODESeW), y una herramienta de edición de servicios, también de la Web Semántica (ODESWS). Para obtener una descripción detallada de cada una de ellas, véase [Gómez-Pérez y otros, 2003].

A continuación, vamos a describir el editor de ontologías de WebODE. Este editor es una aplicación Web que se ha construido sobre el servicio de acceso de la ontología

⁶<http://webode.dia.fi.upm.es/>

(ODE API), y que para construir ontologías en la plataforma integra varios servicios: edición de la ontología, navegación, documentación, mezcla, razonamiento, etc. En este editor se combinan tres interfaces de usuario, a saber: un editor HTML basado en formularios para editar todos los términos ontológicos menos los axiomas y las reglas, una interfaz gráfica de usuario, llamada ODEDesigner, para editar gráficamente taxonomías de conceptos y relaciones, y WAB (*WebODE Axiom Builder*) para editar axiomas y reglas formales. En esta sección vamos a describir dichas interfaces destacando sus rasgos más importantes.

La Figura 5.7 muestra una captura de pantalla de la interfaz HTML para editar atributos de instancias del concepto **persona física** de nuestra ontología. Las principales áreas de este interfaz son:

- El área de exploración. Para navegar por toda la ontología, crear nuevos elementos y modificar o borrar los que ya existen.
- El portapapeles. Para copiar y pegar fácilmente información entre formularios, y así crear de una forma fácil componentes similares de la ontología.
- El área de edición. Para introducir, borrar y actualizar términos de la ontología (conceptos, atributos, relaciones, etc.) con formularios HTML y tablas con conocimiento sobre los términos existentes.

La Figura 5.7 muestra los atributos definidos para el concepto persona física, que son: **edad**, **fecha de nacimiento**, **fecha de muerte**, etc.

ODEDesigner facilita la construcción de taxonomías de conceptos y de relaciones binarias *ad hoc* entre conceptos, a la vez que permite definir vistas para destacar o personalizar la visualización de fragmentos de la ontología a distintos usuarios.

Las taxonomías de conceptos se crean con las siguientes relaciones conjuntos de relaciones predefinidas: *subclase de*, *descomposición disjunta*, *descomposición exhaustiva*, *partición* y *parte de*. La Figura 5.5 y la Figura 5.6 muestran distintas vistas de nuestra ontología de entidades legales, hechas en ODEDesigner.

Instance Attributes for Term **physical person**.

Attribute	Description	Type	Cardinality	Placeholder	Value	Value Type
age	The date when a person was born	Date	[1..1]	now	0	
birthDate	The date when a person was born	Date	[1..1]			
deathDate	The date when a person died	Date	[1..1]			
name	Name	String	[1..1]			
nationality	Nationality	String	[1..1]			
sex	Sex	String	[1..1]			

New Attribute
Attribute Attribute Name:
Description:
Type:

Figura 5.7: Edición de un atributo de instancia con el editor de ontologías WebODE.

El *WebODE Axiom Builder* (WAB) es un editor gráfico para construir axiomas formales y reglas como las que aparecen en la Tabla 5.7 y la Tabla 5.8. Dicho editor

tiene como objetivo facilitar a los expertos en el dominio, y que no tengan mucha experiencia en modelar en lógica de primer orden, la creación de los componentes antes comentados.

A continuación vamos a describir otros servicios para construir ontologías que se encuentran integrados en el editor de la ontología, a saber: el servicio de documentación, ODEMerge, y el servicio de evaluación. Hay muchos otros servicios que no vamos a describir aquí porque creemos que no tienen ninguna utilidad especial para los lectores a los que este libro va dirigido, algunos de estos son: el motor de inferencias de Prolog que está basado en OKBC, ODEClean, los servicios de traducción etc.

El servicio de documentación de ontologías de WebODE genera documentación de ontologías WebODE en diferentes formatos, como son las tablas en HTML que muestran las representaciones intermedias de METHONTOLOGY, descritas en la sección 5.3, y las taxonomías de conceptos en HTML.

El servicio de mezcla de WebODE (ODEMerge) realiza y supervisa la fusión de conceptos, atributos y relaciones binarias *ad hoc* pertenecientes a dos ontologías construidas para el mismo dominio; este servicio utiliza fuentes de lenguaje natural para encontrar correspondencias entre los componentes de las dos ontologías y así generar una nueva ontología. Finalmente, comentaremos que la plataforma WebODE, ofrece una serie de funciones para evaluar ontologías tales como la que analiza la consistencia de las ontologías o los servicios de evaluación de RDF(S), DAML+OIL, y de OWL.

El servicio que asegura la consistencia de las ontologías ofrece la posibilidad de controlar las restricciones de las ontologías de WebODE y lo utiliza el editor durante el proceso de construcción. Dicho servicio controla las restricciones de tipo, de los valores numéricos y de cardinalidad y verifica las taxonomías de conceptos (por ejemplo, instancias externas de descomposición exhaustiva, bucles, etc.)

Los servicios de evaluación de RDF(S), DAML+OIL y OWL evalúan ontologías siguiendo los criterios de evaluación señalados por [Gómez-Pérez, 2006], y detectan los errores de las ontologías implementadas en los susodichos lenguajes a la vez que sugieren unos criterios mejores para el diseño de las ontologías.

5.5 Otros métodos y herramientas para desarrollar ontologías

En esta sección vamos a explicar los métodos, la metodología y las herramientas que no hemos comentado en los capítulos anteriores.

5.5.1 Metodologías y métodos

Existen en la literatura y desde hace tiempo, una serie de métodos y metodologías para desarrollar ontologías. Ya en 1990, [Lenat y Guha, 1990] publicaron una serie de guías generales e hicieron algunos comentarios interesantes sobre el desarrollo de Cyc; años después, en 1995, [Grüninger y Fox, 1995], propusieron las primeras directrices basándose en la experiencia que habían adquirido al desarrollar la *Enterprise Ontology* [Uschold y Grüninger, 1996] y el proyecto de ontologías TOVE (*TOronto Virtual*

Enterprise) ambas ontologías pertenecientes al dominio de modelado de la empresa, [Grüninger y Fox, 1995], que fueron más tarde mejoradas en [Uschold y Grüninger, 1996].

En la XII Conferencia Europea de Inteligencia Artificial (ECAI'96) [Bernaras y otros, 1996] presentaron, como parte del proyecto Esprit KACTUS, el método utilizado para construir una ontología en el dominio de las redes eléctricas; la metodología METHONTOLOGY apareció simultáneamente a aquella y ha seguido ampliándose en artículos sucesivos [Corcho y otros, 2007; Fernández-López y otros, 1997, 1999]. En 1997, se propuso un nuevo método para construir ontologías, basado en la ontología SENSUS, [Benton y Kambhampati, 1997] y posteriormente, apareció la metodología On-To-Knowledge , desarrollada dentro del proyecto del mismo nombre [Staab y otros, 2001].

Dependiendo de los objetivos marcados, podemos utilizar una metodología u otra. METHONTOLOGY es la metodología que describe de forma más detallada los procesos a realizar. Por otro lado, On-To-Knowledge es la que cubre mayor número de actividades aunque con muy pocas descripciones de los procesos llevados a cabo; en cuanto a la metodología de Grüninger y Fox debemos comentar que es la más formal de todas y, aunque todas las metodologías cuentan con reutilizar las ontologías existentes durante el proceso de desarrollo, sólo METHONTOLOGY ha adaptado, recientemente, su propuesta de un ciclo de vida al entorno de las ontologías en red. De todas formas, hemos de comentar que las actividades de desarrollo, y sobre todo las de conceptualización e implementación son las que aparecen de forma más detallada en todas las metodologías. Sin embargo, todavía se echan en falta propuestas para actividades de gestión (planificación, control y garantía de calidad), así como de actividades previas al desarrollo (estudio del entorno) y posteriores al desarrollo (por ejemplo, la [re]utilización).

Para ampliar el estudio sobre las metodologías para el desarrollo de ontologías se puede consultar [Fernández-López y Gómez-Pérez, 2002].

5.5.2 Herramientas

En cuanto a la tecnología de las herramientas, podemos comentar que ésta ha mejorado enormemente desde la creación de los primeros entornos. Si tenemos en cuenta su evolución desde su aparición a mediados de los noventa, podemos distinguir dos grupos distintos:

- Herramientas cuyo modelo de conocimiento se corresponde directamente con un lenguaje de implementación de ontologías. Estas herramientas se desarrollaron como editores de ontologías para un lenguaje específico y dentro de este grupo podemos incluir a Ontolingua Server [Farquhar y otros, 1997], que respalda la construcción de ontologías con Ontolingua y con KIF; a Ontosaurus Server [Benton y Kambhampati, 1997] con Loom, y a OilEd [Bechhofer y otros, 2001], primero con OIL, luego con DAML+OIL y ahora con OWL.
- Juegos de herramientas integradas cuya principal característica es que tienen una arquitectura extensible y su modelo de conocimientos es generalmente in-

dependiente de un lenguaje de ontologías. Además, ofrecen un conjunto básico de servicios relacionados con las ontologías y pueden ampliarse fácilmente con otros módulos para ofrecer más funciones. A este grupo pertenecen Protégé-2000 [Noy y otros, 2000]), WebODE [Arpírez y otros, 2003], OntoEdit [Benton y Kambhampati, 2002] y KAON [Maedche y otros, 2003].

Protégé [Noy y otros, 2000]. Esta herramienta ha sido desarrollada en el *Stanford Medical Informatics* (SMI) de la Universidad de Stanford. Es una aplicación autónoma, de código abierto y con arquitectura extensible. El núcleo de este entorno es el editor y la herramienta tiene una biblioteca de extensiones que le da más funcionalidad al entorno. Normalmente, las extensiones o plug-ins están disponibles para importar/exportar lenguajes de implementación de ontologías (FLogic, Jess, XML, Prolog), diseño de lenguajes, [Knublauch y otros, 2004], acceso a OKBC, creación y ejecución de restricciones (PAL), fusión de ontologías [Noy y Musen, 2000], etc.

OntoEdit [Benton y Kambhampati, 2002]. Es una herramienta desarrollada en la Universidad de Karlsruhe y comercializada por Ontoprise. Es similar a las anteriores, es decir, es un entorno extensible y flexible, basado en una arquitectura de extensiones que ofrece cierta funcionalidad para explorar y editar ontologías, además, incorpora extensiones que se encargan de hacer inferencias utilizando Ontobroker y de exportar e importar ontologías en diferentes formatos (FLogic, XML, RDF(S) y OWL), etc. Hay dos versiones de OntoEdit disponibles: la gratuita, Ontoedit Free, y la profesional, Ontoedit Professional.

El juego de herramientas de **KAON1** [Maedche y otros, 2003] es un entorno de ingeniería ontológica, de código abierto y extensible que define el modelo de conocimientos que subyace en una extensión de RDF(S). OI-modeler es el editor de estas herramientas y el encargado de hacer evolucionar las ontologías, encontrar correspondencias entre ellas y generarlas a partir de las bases de datos, etc.

El IBM Ontology Management System, también conocido por **SNOBASE**, carga ontologías desde ficheros a través de Internet para crear, modificar, consultar y almacenar ontologías localmente. Permite manipular ontologías escritas en RDF Schema y OWL. En la actualidad, SNOBASE respalda una variante de OWL Query Language (OWL-QL) [Fikes y otros, 2003].

Swoop [Kalyanpur y otros, 2006] es una herramienta basada en una arquitectura de extensiones, que respalda la anotación en colaboración a través de Annotea [Koivunen, 2006], y detecta las inconsistencias. Actualmente se trabaja para que Swoop dé soporte a la evolución de ontologías.

Un aspecto interesante de las herramientas es que solamente OntoEdit y WebODE **respaldan metodologías para construir ontologías** (On-To-Knowledge y MET-HONTOLOGY respectivamente), aunque esto no les impida ser utilizadas con otras metodologías o con ninguna.

Desde la perspectiva del paradigma de representación de conocimientos, las herramientas KAON1 se basan en redes semánticas y en marcos. Por otro lado, WebODE, Protégé, OntoEdit y KAON1 pueden representar conocimientos de acuerdo con una propuesta híbrida que se basa en marcos y en lógica de primer orden. En cuanto a SNOBASE y Swoop, éstas utilizan lógicas descriptivas. Un hecho a tener en cuenta es

la expresividad del modelo de conocimientos subyacente en la herramienta. Con todas las herramientas se pueden representar clases, relaciones, atributos e instancias pero sólo KAONI y Protégé cuentan con componentes flexibles como las metaclasses. También es importante conocer, antes de escoger una herramienta para desarrollar ontologías, los servicios de inferencias adjuntos a la herramienta como: mecanismos para examinar las restricciones y la consistencia, el tipo de herencia (sencilla, múltiple, monotónica, no-monotónica) clasificaciones automáticas, manejo de las excepciones y ejecución de los procedimientos. KAONI no tiene motor de inferencias, OntoEdit utiliza FLogic [Kifer y otros, 1995] como motor de inferencias; WenODE utiliza Ciao Prolog [Hermenegildo y otros, 2000], Protégé utiliza un motor interno PAL, con SNOBASE se pueden hacer consultas a través de OWL-Ql, y Swoop utiliza un motor de inferencias basado razonadores de lógica de primer orden (por defecto, Pellet). Además Protégé y WebODE ofrecen servicios para evaluar las ontologías y ambos llevan un módulo que realiza las evaluaciones según el método OntoClean [Guarino y Welty, 2002; Welty y Guarino, 2001]. Por último, Protégé (con las extensiones de OWL) realiza clasificaciones automáticas al conectarse a un razonador de lógicas descriptivas.

Otros aspectos importantes de las herramientas a tener en cuenta son **la arquitectura de software y la evaluación de herramientas** (auténtoma, cliente/servidor, aplicación de n-hileras/filas) extensibilidad, almacenamiento de las ontologías (bases de datos, ASCII, ficheros, etc.), tolerancia a fallos, gestión de seguridad/copias/respaldo, estabilidad y política sobre las versiones de las herramientas. Desde esta perspectiva, todas estas herramientas basadas en plataformas Java pueden servir para almacenar ontologías en bases de datos. La función para la gestión de seguridad la ofrece WebODE, mientras que KAON, OntoEdit, Protégé, WebODE, SNOBASE y Swoop ofrecen servicios de extensibilidad.

La **interoperabilidad** con otras herramientas, con sistemas de información y con bases de datos, así como la traducción a y desde algunos lenguajes de ontologías son otros rasgos importantes a tener en cuenta al integrar ontologías en aplicaciones. Casi todas las herramientas importan y exportan XML y otros lenguajes de demarcación, y aunque todavía no existe un estudio que compare la calidad de todos estos traductores, tampoco existen resultados empíricos sobre la posibilidad de intercambiar ontologías entre diferentes herramientas ni sobre la cantidad de conocimientos que se pierden en los procesos de traducción. En el taller de EON2003 se dieron a conocer algunos de los trabajos que se están realizando sobre el tema.

En cuanto a **cooperar y colaborar en la construcción de ontologías**, Protégé incorpora algunas funcionalidades de sincronización, Por lo general, las herramientas existentes necesitan tener casi todas las características antes aludidas para que la colaboración en la construcción de ontologías tenga éxito. XML.com tiene un inventario de herramientas para el desarrollo de ontologías⁷.

⁷<http://www.xml.com/pub/a/2004/07/14/onto.html>

5.6 Lenguajes de implementación de ontologías

Los lenguajes de implementación de ontologías se comenzaron a elaborar a principios de los años noventa, y fueron el resultado de la evolución normal de los lenguajes de representación de conocimiento (KR). Básicamente, los paradigmas de representación de conocimientos subyacentes a los lenguajes de ontologías estaban basados en lógica de primer orden (p.ej., KIF [Genesereth y Fikes, 1992], en marcos combinados con dicha lógica, (p.ej., Ontolingua [Farquhar y otros, 1997; Gruber, 1992], OCML [Motta, 1999] y FLogic [Kifer y otros, 1995]), y en lógicas descriptivas (p.ej., Loom [MacGregor, 1991]). En 1997, se creó OKBC [Chaudhri y otros, 1998] para que actuara como un protocolo unificador basado en marcos y acceder a las ontologías implementadas en diferentes lenguajes (Ontolingua, Loom y CycL, entre otras), si bien sólo se utilizó en un reducido número de aplicaciones.

El gran auge de Internet hizo que se crearan lenguajes de implementación de ontologías para poder explotar las características de la Web; a estos lenguajes se les conoce normalmente como lenguajes de ontologías basados en la Web o lenguajes de marcación de las ontologías y su sintaxis se fundamenta en la sintaxis de los lenguajes de marcación existentes como HTML [Raggett y otros, 1999] y XML [Bray y otros, 2000], cuyo objetivo no es el desarrollo de ontologías sino la presentación y el intercambio de datos, respectivamente. Los ejemplos más sobresalientes de dichos lenguajes de demarcación son: SHOE [Luke y Heflin, 2000], XOL [Karp y otros, 1999], RDF [Lassila y Swick, 2000], RDF Schema [Brickley y Guha, 2004], OIL [Horrocks y otros, 2000], DAML+OIL [van Harmelen y otros, 2001] y OWL [Dean y Schreiber, 2004], y de todos ellos sólo RDF, RDF Schema y OWL están recibiendo respaldo de forma activa. Por último, debemos añadir que se está desarrollando una ontología, llamada WSML en el contexto del trabajo que se realiza en el campo de la Web Semántica y más concretamente en el marco del WSMO. A continuación, vamos a comentar los lenguajes de marcado más importantes ya que son muy útiles dentro del contexto de los Servicios Web Semánticos.

RDF [Lassila y Swick, 2000] fue desarrollado por el W3C como un lenguaje basado en la red semántica para describir recursos Web. También RDF Schema [Brickley y Guha, 2004] fue elaborado por el W3C como una extensión de RDF con primitivas basadas en marcos. La combinación de ambos lenguajes, RDF y RDF Schema, RDF(S), y este lenguaje, para el que se han creado motores de inferencia y lenguajes de consulta, sólo sirve para representar conceptos, taxonomías de conceptos y relaciones binarias.

OWL (Ontology Web Language) fue recomendado por el W3C en febrero del 2004. Este lenguaje está construido encima de RDF(S) y amplía su expresividad con más primitivas, lo que le permite representar expresiones complejas y describir conceptos y relaciones. OWL está dividido en tres capas o estratos (Lite, DL y Full), cada una de las cuales ofrece distintos niveles de expresividad según las necesidades de representación e inferencia de la ontología. Este lenguaje se basa en SHOIN(D+), un lenguaje de lógica descriptiva que cuenta con varios motores de inferencia que se pueden utilizar para examinar las restricciones de conceptos, propiedades e instancias y para clasificar, automáticamente, los conceptos de forma jerárquica.

Por ejemplo, con OWL podemos describir un vuelo como una clase de viaje en el que el medio de transporte utilizado es un avión. Si determinamos esta condición como necesaria y suficiente y luego definimos un viaje en el que se utiliza un avión ligero como medio de transporte (y asumimos que *avión ligero* es una especialización de *avión*), entonces el razonador concluirá que este *viaje* es una especialización de *vuelo*. Este mismo principio se puede utilizar igualmente para examinar la consistencia de la definición que la ontología ofrece.

Y por último, el lenguaje **WSML** (Web Service Modelling Language) [Bruijn, 2006] que se está desarrollando en el marco del WSMO. El objetivo de este lenguaje es que sea utilizado no únicamente en la representación de ontologías sino también en la de los Servicios Web Semánticos, para lo cual cuenta con muchas características adicionales que los lenguajes antes mencionados no tienen. Al igual que OWL, está dividido en capas, cada una de las cuales se basa en distintos formalismos KR, a saber: lógica descriptiva, programación lógica y lógica de primer orden.

De todos estos lenguajes aquí comentados, sólo algunos como Ontolingua, OCML, Flogic, RDF, RDF Schema y OWL (OIL y DAML+OIL también respaldan esta noción pero ya no están activos), están bien equipados con primitivas para explotar el concepto de ontologías en red. Basándonos en nuestra experiencia y en los estudios de los casos disponibles en la literatura, hemos podido encontrar algunas asociaciones entre los lenguajes de ontología y las diferentes clases de aplicaciones basadas en ontologías en las que se aplican.

En las aplicaciones de comercio electrónico, las ontologías se utilizan generalmente para representar productos que se ofrecen en plataformas electrónicas y se muestran a los usuarios a través de catálogos que pueden examinar [Léger y otros, 2000]. Las exigencias de representación de estas ontologías no son extremas, básicamente se necesitan conceptos y atributos y relaciones n-arias entre conceptos; sin embargo, las exigencias de razonamiento son mayores: si el número de productos o servicios que se ofrecen en la plataforma es alto, entonces las clasificaciones automáticas son muy útiles para organizar los productos o servicios (por lo que los lenguajes basados en lógica de descripción son muy, muy útiles) e igual de útil es una eficiente contestación a las consultas (esta rasgo lo tienen casi todos los lenguajes estudiados).

Cuando se utilizan conjuntamente PSMs y ontologías de dominio, hay dos lenguajes que se recomiendan encarecidamente: OCML y FLogic [Fensel, 1995; Motta, 1999], porque respaldan explícitamente la integración y la reutilización de bibliotecas.

Dentro del contexto de la Web Semántica y para poder intercambiar ontologías entre aplicaciones, hay una serie de lenguajes basados en XML que son fáciles de leer y gestionar porque las bibliotecas estándares para el tratamiento de XML están disponibles gratuitamente. Además, no es difícil adaptar los lenguajes tradicionales a la sintaxis de XML y así podrían utilizar el mismo tipo de bibliotecas. El gran mérito de RDF(S) y OWL es el amplio apoyo que reciben de comunidades ajenas a la comunidad ontológica, lo que implica que haya más herramientas disponibles para editar, gestionar y documentar ontologías.

Crear ontologías de nivel superior exige una gran expresividad pero no requiere gran respaldo en el razonamiento. Estas ontologías están normalmente especificadas en lenguajes de lógicas descriptivas como LOOM y CLASSIC.

Por lo general, los lenguajes que se basan en lógicas descriptivas han sido muy utilizados en aplicaciones que requieren integración inteligente o fuentes de información heterogéneas y recogida de información [Barish y otros, 2000; Guarino y otros, 2006; Mena y otros, 1996; Stuckenschmidt, 2000]. Y la razón de que sean tan utilizadas se debe a su soporte de inferencia.

5.7 Resumen

A comienzos de los años noventa, el desarrollo de las ontologías era un arte: los desarrolladores no tenían unas directrices claras para construirlas, sólo contaban con algunos criterios de diseño a seguir. El trabajo que se ha ido haciendo sobre principios, métodos y metodologías además de la tecnología existente ha convertido el desarrollo de ontologías en una ingeniería. Este proceso migratorio se debe principalmente a la definición del proceso de desarrollo de ontologías y de su ciclo de vida, que describe los pasos a seguir para construir ontologías y las interdependencias entre los pasos.

En este capítulo hemos estudiado los componentes de las ontologías, los métodos, metodologías, herramientas y lenguajes que existen en la actualidad y además hemos mostrado con un ejemplo cómo se puede desarrollar una ontología utilizando una metodología y un software en particular: METHONTOLOGY y WebODE. En concreto, hemos mostrado cómo se desarrolla una ontología de entidades legales en el contexto español al adaptar una taxonomía, también de entidades legales, creada por Breuker. Una conclusión evidente que podemos extraer de este ejemplo es que no es necesario tener una gran experiencia en la representación del conocimiento para hacer una ontología. Con METHONTOLOGY podemos modelar ontologías a través de representaciones intermedias tanto gráficas como tabulares que los expertos de cualquier dominio comprenden sin estar necesariamente involucrados en el mundo de las ontologías. Además, WebODE es una plataforma de software que da apoyo a METHONTOLOGY, aunque no fuerza a seguir dicha metodología.

En este capítulo también hemos presentado otros métodos y herramientas para que los lectores puedan trabajar con otras propuestas.

En cuanto a las herramientas, hay que decir que sólo hemos descrito brevemente las que consideramos más relevantes en la actualidad. Curiosamente, sólo WebODE y OntoEdit están pensadas para poder dar soporte a las metodologías METHONTOLOGY y OntoKnowledge, aunque no sólo a ellas. Y añadir que hoy por hoy, el que una herramienta deba ser el reflejo de una metodología no es una idea muy generalizada.

Los lenguajes de ontologías han estado siempre en continua evolución desde que los primeros lenguajes estaban disponibles para implementar ontologías, la mayoría de los cuales estaban fundamentados en los lenguajes que existían de representación de conocimientos.

En los recientes avances habidos en el desarrollo de lenguajes, creados en el contexto de la Web Semántica (RDF, RDF Schema and OWL) y de los Servicios Web Semánticos (WSML), se han tenido muy en cuenta las ontologías en red discrepantes, además, se han añadido espacios de nombres que designan componentes de ontologías definidos en cualquier otro sitio, y se utilizan las primitivas de importación con objeto

de incluir ontologías ya existentes. A continuación, mostramos algunos enlaces a las ontologías más relevantes:

• **ONTOLOGÍAS DE ALTO NIVEL:**

- DOLCE (<http://www.loa-cnr.it/DOLCE.html>)
- SUO (<http://suo.ieee.org/>)

• **ONTOLOGÍAS LINGÜÍSTICAS**

- WordNet (wordnet.princeton.edu/)
- EuroWordnet (<http://www.hum.uva.nl/ewn/>)
- SENSUS (<http://www.hum.uva.nl/ewn/>)

• **ONTOLOGÍAS DE COMERCIO ELECTRÓNICO**

- UNSPSC (<http://www.unpsc.org/>)
- RosettaNet (<http://www.rosettanet.org/>)

• **ONTOLOGÍAS MÉDICAS**

- UMLS (<http://www.nih.gov/research/umls/>)
- GALEN (<http://www.opengalen.org/>)

• **ONTOLOGÍAS DE BIOINFORMÁTICA**

- GeneOntology (<http://www.geneontology.org/>)

5.8 Ejercicios resueltos

5.1. ¿Qué cambiaría y añadiría a la ontología de la Figura 5.8?

Respuesta:

- Supercomputador no es parte de ordenador, sino una subclase.
- No es recomendable omitir partes de los nombres, por ejemplo, en vez de “portátil”, el nombre del concepto debería ser “ordenador portátil”.
- La ontología está reflejando una situación demasiado particular, en la que los portátiles son negros, y los PCs y los supercomputadores, negros.
- La ontología no está reflejando los aspectos esenciales de los conceptos que se están describiendo. Por ejemplo, las características del procesador, de la memoria, etc, son importantes, y no aparecen.
- El primer axioma de la ontología es una tautología.
- Hay una contradicción entre los otros dos axiomas. Mientras que el primero dice que un ordenador portátil tiene que pesar menos de 6 kg el segundo afirma que existe un portátil de 9 kg.

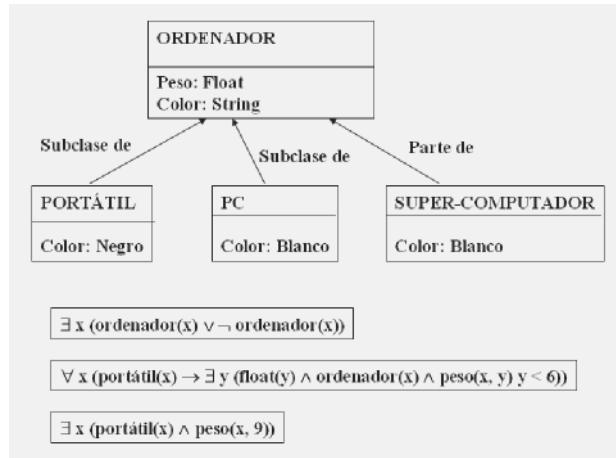


Figura 5.8: Ontología a evaluar.

- En caso de que se desee extender la ontología, hay varias alternativas (no excluyentes), por ejemplo:
 1. Creando ontologías sobre componentes de ordenadores.
 2. Integrando alguna ontología de unidades de medida para que el peso, por ejemplo, no tenga que estar expresado como un float, sino como un peso, concepto definido en otra ontología.
 3. Integrando alguna ontología de colores. Esta opción sólo es adecuada si el color se va a considerar una característica importante.
 4. Integrando alguna ontología con conceptos abstractos (objeto físico, característica, etc.) para afinar más el significado de los términos de la ontología de ordenadores.

■

5.9 Ejercicios propuestos

5.1. Utilizando WebODE, conceptuar, según METHONTOLOGY, una ontología de coches y después implementarla en OWL. Definir los términos que aparecen a continuación:

- Car.
- Sports car.
- Van.
- Berline.

- Horsepower.
- Maximum speed.
- Gasoline consumption.
- Type of brakes.
- Dimensions (length, high, etc.)

5.2. Importar, de nuevo con WebODE, la ontología OWL construida en el ejercicio 1. ¿Se ha perdido algún conocimiento? En caso negativo, ¿hay algún caso en que se pudiera haber perdido?

5.3. Cargar en Protégé la ontología OWL elaborada en el ejercicio 1. ¿Ha tenido algún problema?

5.4. Tomar cualquiera de las ontologías de Protégé e intente importarla desde WebODE.

5.5. ¿Cómo podría utilizar la ontología del ejercicio 1 para desarrollar un sistema Web que ayude a la compra-venta de vehículos?

Referencias

- ARPÍREZ, J.C.; CORCHO, O.; FERNÁNDEZ-LÓPEZ, M. y GÓMEZ-PÉREZ, A.: «WebODE in a nutshell». *AI Magazine*, 2003, **24(3)**, pp. 37–47.
- BARISH, G.; KNOBLOCK, C.A.; CHEN, Y.S.; MINTON, S.; PHILPOT, A. y SHAHABI, C.: «The theaterloc virtual application». En: R. Engelmore y H. Hirsh (Eds.), *Proceedings of the 12th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pp. 980–987. Austin, Texas, 2000.
- BECHHOFER, S.; HORROCKS, I.; GOBLE, C. y STEVENS, R.: «OilEd: a reasonable ontology editor for the Semantic Web». En: F. Baader; G. Brewka y T. Eiter (Eds.), *Proceedings of the oint German/Austrian conference on Artificial Intelligence (KI?01) (Lecture Notes in Artificial Intelligence LNAI 2174)*, pp. 396–408. Springer-Verlag, Berlin, Germany, Vienna, Austria., 2001.
- BENTON, J. y KAMBHAMPTI, S.: «Toward Distributed Use of Large-Scale Ontologies». En: A. Farquhar; M. Gruninger; A. Gómez-Pérez; M. Uschold y P. van der Vet (Eds.), *Proceedings of the AAAI'97 Spring Symposium on Ontological Engineering*, pp. 138–148. Stanford University, California, 1997.
- BENTON, J. y KAMBHAMPTI, S.: «OntoEdit: Collaborative Ontology Engineering for the Semantic Web». En: I. Horrocks y J.A. Hendler (Eds.), *Proceedings of the First International Semantic Web Conference (ISWC'02) (Lecture Notes in Computer Science LNCS 2342)*, pp. 221–235. Springer-Verlag, Berlin, Germany, Sardinia, Italy, 2002.
- BERNARAS, A.; LARESGOITI, I. y CORERA, J.: «Building and reusing ontologies for electrical network applications». En: W. Wahlster (Ed.), *Proceedings of the European Conference on Artificial Intelligence (ECAI'96)*, pp. 298–302. John Wiley and Sons, Chichester, United Kingdom, Budapest, Hungary, 1996.
- BERNERS-LEE, T.: *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. HarperCollins Publishers, New York, 1999.
- BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C.M. y MALER, E.: «Extensible Markup Language (XML) 1.0. W3C Recommendation». *Informe técnico*, W3C, 2000. Disponible en <http://www.w3.org/TR/PR-rdf-schema>.
- BRICKLEY, D. y GUHA, R.V.: «RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation». *Informe técnico*, W3C, 2004. Disponible en <http://www.w3.org/TR/PR-rdf-schema>.
- BRUIJN, J. DE: «The Web Service Modeling Language WSML. Deliverable D16.1v0.21.» *Informe técnico*, ESSI WSML working group, 2006. Disponible en <http://www.wsmo.org/TR/d16/d16.1/v0.21/>.

- CHAUDHRI, VK; FARQUHAR, A; FIKES, R; KARP, PD y RICE, JP: «Open Knowledge Base Connectivity 2.0.3». *Informe técnico*, Artificial Intelligence Center. SRI International, 1998. Disponible en <http://www.ai.sri.com/okbc/okbc-2-0-3.pdf>.
- CHEN, P.P.: «The Entity-Relationship Model: Toward a Unified View of Data». *ACM Transactions on Database Systems*, 1976, **1**(1), pp. 9–36.
- CORCHO, O.; FERNÁNDEZ-LÓPEZ, M. y GÓMEZ-PÉREZ, A.: «Ontological Engineering: Principles, Methods, Tools and Languages». En: C. Calero; F. Ruiz y M. Piattoni (Eds.), *Ontologies for Software Engineering and Technology*, Springer-Verlag, 2007.
- DEAN, M. y SCHREIBER, G.: «OWL Web Ontology Language Reference. W3C Recommendation». *Informe técnico*, W3C, 2004. Disponible en <http://www.w3.org/TR/owl-ref/>.
- DECLERCK, T. y USZKOREIT, H.: «State of the art on multilinguality for ontologies, annotation services and user interfaces. Esperonto deliverable D1.5». *Informe técnico*, Esperonto Project, 2003. Disponible en <http://www.esperonto.net>.
- FARQUHAR, A.; FIKES, R. y RICE, J.: «The Ontolingua Server: A Tool for Collaborative Ontology Construction». *International Journal of Human Computer Studies*, 1997, **6**, pp. 707–727.
- FENSEL, D.: *The knowledge acquisition and representation language KARL*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.
- FERNÁNDEZ-LÓPEZ, M. y GÓMEZ-PÉREZ, A.: «Overview and analysis of methodologies for building ontologies». *The Knowledge Engineering Review*, 2002, **17**(2), pp. 129–156.
- FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. y JURISTO, N.: «METHONTOLOGY: From Ontological Art Towards Ontological Engineering». En: *Proceedings of the Spring Symposium on Ontological Engineering of AAAI*, pp. 33–40. Stanford University, California, 1997.
- FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A.; PAZOS, A. y PAZOS, J.: «Building a Chemical Ontology Using Methontology and the Ontology Design Environment». *IEEE Intelligent Systems & their applications*, 1999, **4**(1), pp. 37–46.
- FIKES, R.; HAYES, P. y HORROCKS, I.: «OWL-QL: A Language for Deductive Query Answering on the Semantic Web». *Informe técnico*, KSL, 2003.
- GANGEMI, A.; PISANELLI, D.M. y STEVE, G.: «An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies». *Data & Knowledge Engineering*, 1999, **31**(2), pp. 183–220.

GENESERETH, M.R. y FIRES, R.E.: «Knowledge Interchange Format. Version 3.0. Reference Manual». *Informe técnico Logic-92-1*, Computer Science Department. Stanford University, California, 1992. Disponible en <http://meta2.stanford.edu/kif/Hypertext/kif-manual.html>.

GRUBER, T.R.: «Ontolingua: A Mechanism to Support Portable Ontologies». *Informe técnico KSL-91-66*, Knowledge Systems Laboratory, Stanford University, Stanford, California, 1992. Disponible en <ftp://ftp.ksl.stanford.edu/pub/KSL-Reports/KSL-91-66.ps>.

GRUBER, T.R.: «A translation approach to portable ontology specification». *Knowledge Acquisition*, 2006, **5(2)**, pp. 199–220.

GRÜNINGER, M. y FOX, M.S.: «Methodology for the design and evaluation of ontologies». En: D. Skuce (Ed.), *Proceedings of the IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing*, pp. 6.1–6.10, 1995.

GUARINO, N.: «Formal Ontology in Information Systems». En: N. Guarino (Ed.), *Proceedings of the 1st International Conference on Formal Ontology in Information Systems (FOIS'98)*, pp. 3–15. IOS Press, Amsterdam, 1998.

GUARINO, N.; MASOLO, C. y VETERE, G.: «OntoSeek: Content-Based Access to the Web». *IEEE Intelligent Systems & their applications*, 2006, **14(3)**, pp. 70–80.

GUARINO, N. y WELTY, C.: «Evaluating Ontological Decisions with OntoClean». *Communications of the ACM* 45(2), 2002, **45(2)**, pp. 61–65.

GÓMEZ-PÉREZ, A.: «Some Ideas and Examples to Evaluate Ontologies». *Informe técnico*, Knowledge Systems Laboratory, Stanford University, California, 1994. Disponible en http://www-ksl.stanford.edu/KSL_Abstracts/KSL-94-65.html.

GÓMEZ-PÉREZ, A.: «Evaluation of Ontologies». *International Journal of Intelligent Systems* 16(3), 2006, **16(3)**, pp. 391–409.

GÓMEZ-PÉREZ, A.; FERNÁNDEZ-LÓPEZ, M. y CORCHO, O.: *Ontological Engineering*. Springer, London, United Kingdom, 2003.

GÓMEZ-PÉREZ, A.; JURISTO, N.; MONTES, C. y PAZOS, J.: *Ingeniería del Conocimiento: Diseño y Construcción de Sistemas Expertos*. Ceura, Madrid, Spain, 1997.

GÓMEZ-PÉREZ, A. y ROJAS, M.D.: «Ontological Reengineering and Reuse». En: D. Fensel y R. Studer (Eds.), *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'99) (Lecture Notes in Artificial Intelligence LNAI 1621)*, pp. 139–156. Springer-Verlag, Berlin, Germany, Dagstuhl Castle, Germany., 1999.

HERMENEGILDO, M.; BUENO, F.; CABEZA, D.; CARRO, M.; GARCÍA, M.; LÓPEZ, P. y PUEBLA, G.: «The Ciao Logic Programming Environment». En: J.W. Lloyd; V. Dahl; U. Furbach; M. Kerber; K. Lau; C. Palamidessi; L.M. Pereira; Y. Sagiv y

- P.J. Stuckey (Eds.), *Proceedings of the International Conference on Computational Logic (CL'00) (Lecture Notes in Computer Science LNCS 1861)*, Springer-Verlag, Berlin, Germany, London, United Kingdom, 2000.
- HORROCKS, I.; FENSEL, D.; VAN HARMELEN, F.; DECKER, S.; ERDMANN, M. y KLEIN, M.: «OIL in a Nutshell». En: R. Dieng y O. Corby (Eds.), *Proceedings of the 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00) (Lecture Notes in Artificial Intelligence LNAI 1937)*, pp. 1–16. Springer-Verlag, Berlin, Germany, Juan-Les-Pins, France, 2000.
- IEEE: *Domain-Independent Temporal Planning in a Planning-Graph-Based Approach*. IEEE Computer Society, New York, 1996.
- KALYANPUR, A.; PARSIA, B.; SIRIN, E.; GRAU, B.C. y HENDLER, J.: «Swoop: A Web Ontology Editing Browser». *Web Semantics: Science, Services and Agents on the World Wide Web*, 2006, **4**(2), pp. 144–153.
- KARP, P.D.; CHAUDHRI, V. y THOMERE, J.: «XOL: An XML-Based Ontology Exchange Language. Version 0.3». *Informe técnico*, Artificial Intelligence Center. SRI International, 1999. Disponible en <http://www.ai.sri.com/pkarp/xol/xol.html>.
- KIETZ, J.U.; MAEDCHE, A. y VOLZ, R.: «A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet». En: N. Aussenac-Gilles; B. Biébow y S. Szulman (Eds.), *Proceedings of the EKAW'00 Workshop on Ontologies and Texts. CEUR Workshop Proceedings 51*, pp. 4.1–4.14. Juan-Les-Pins, Franc, 2000. Disponible en <http://CEUR-WS.org/Vol-51/>.
- KIFER, M.; LAUSEN, G. y WU, J.: «Logical Foundations of Object-Oriented and Frame-Based Languages». *Journal of the ACM*, 1995, **42**(4), pp. 741–843.
- KNUBLAUCH, H.; FERGERSON, R.; NOY, N.F. y MUSEN, M.A.: «The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications». En: S.A. McIlraith y D. Plexousakis (Eds.), *Proceedings of the 3rd International Semantic Web Conference (ISWC'04) (Lecture Notes in Computer Science LNCS 3298)*, pp. 229–243. Springer-Verlag, Berlin, Germany, Hiroshima, Japan, 2004.
- KOIVUNEN, M.R.: «Web Tagging with Annotea Shared/Social Bookmarks and Topics». En: *Proceedings of the Tagging Workshop at the World Wide Web Conference*, pp. 23–25. Edinburgh, United Kingdom, 2006. Disponible en <http://annotea.org/www2006/annotea.htm>.
- LASSILA, O. y SWICK, R.: «Resource Description Framework (RDF) Model and Syntax Specification (W3C Recommendation)». *Informe técnico*, W3C, 2000. Disponible en <http://www.w3.org/TR/REC-rdf-syntax/>.
- LENAT, D.B. y GUHA, R.V: *Building Large Knowledge-Based Systems*. Addison-Wesley, 1990.

- LOWE, E.J.: *The Four-Category Ontology: A Metaphysical Foundation for Natural Science*. Oxford University Press, 2006.
- LUKE, S y HEFLIN, JD: «SHOE 1.01. Proposed Specification». *Informe técnico*, Parallel Understanding Systems Group. Department of Computer Science. University of Maryland, 2000. Disponible en <http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm>.
- LÉGER, A.; ARBANT, G.; BARRETT, P.; GITTON, S.; GÓMEZ-PÉREZ, A.; HOLM, R.; LEHTOLA, A.; MOUGENOT, I.; NISTAL, A.; VARVARIGOU, T. y VINESSE, J.: «MKBEEM: Ontology domain modeling support for multilingual services in e-Commerce». En: *Proceedings of the ECAI'00 Workshop on Applications of Ontologies and PSMs*, pp. 19.1–19.4. Berlin, Germany, 2000.
- MACGREGOR, R.: «Inside the LOOM clasifier». *SIGART bulletin*, 1991, **2(3)**, pp. 70–76.
- MAEDCHE, A.; MOTIK, B.; STOJANOVIC, L.; STUDER, R. y VOLZ, R.: «Ontologies for Enterprise Knowledge Management». *IEEE Intelligent Systems*, 2003, **18(2)**, pp. 26–33.
- MENA, E.; KASHYAP, V.; SHETH, A.P. y ILLARRAMENDI, A.: «OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies». En: W. Litwin (Ed.), *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pp. 14–25. Brussels, Belgium, 1996.
- MOTTA, E.: *Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design*. IOS Press, Amsterdam, The Netherlands, 1999.
- NECHES, R.; FIKES, R.E.; FININ, T.; GRUBER, T.R.; SENATOR, T. y SWARTOUT, W.R.: «Enabling technology for knowledge sharing». *AI Magazine*, 1991, **12(3)**, pp. 36–56.
- NOY, N.F.: «Evaluation by Ontology Consumers». *IEEE Intelligence Systems*, 2006, **19(4)**, pp. 80–81.
- NOY, N.F.; FERGERSON, R.W. y MUSEN, M.A.: «The knowledge model of Protege-2000: Combining interoperability and flexibility». En: R. Dieng y O. Corby (Eds.), *Proceedings of the 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00) (Lecture Notes in Artificial Intelligence LNAI 1937)*, pp. 17–32. Springer-Verlag, Berlin, Germany, Juan-Les-Pins, France, 2000.
- NOY, N.F. y MUSEN, M.A.: «PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment». En: P. Rosenbloom; H.A. Kautz; B. Porter; R. Dechter; R. Sutton y V. Mittal (Eds.), *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, pp. 450–455. Austin, Texas, 2000.

- OMG: «Ontology Definition Metamodel. Third Revised Submission to OMG/ RFP ad/2003-03-40». *Informe técnico*, Object Management Group, 2005. Disponible en <http://www.omg.org/docs/ad/05-08-01.pdf>.
- RAGGETT, D.; HORS, A. LE y JACOBS, I.: «HTML 4.01 Specification. W3C Recommendation». *Informe técnico*, W3C, 1999. Disponible en <http://www.w3.org/TR/html401/>.
- RUMBAUGH, J.; JACOBSON, I. y BOOCHE, G.: *The Unified Modeling Language Reference Manual*. Addison-Wesley, Boston, Massachusetts, 1998.
- STAAB, S.; SCHNURR, H.P.; STUDER, R. y SURE, Y.: «Knowledge Processes and Ontologies». *IEEE Intelligent Systems*, 2001, **16(1)**, pp. 26–34.
- STEVE, G.; GANGEMI, A. y PISANELLI, D.M.: «Integrating Medical Terminologies with ONIONS Methodology». En: Kangassalo H, Charrel JP (eds) *Information Modeling and Knowledge Bases VIII*, IOS Press, Amsterdam, The Netherlands, 1998. <Http://ontology.ip.rm.cnr.it/Papers/onions97.pdf>.
- STUCKENSCHMIDT, H.: «Using OIL for Intelligent Information Integration». En: V.R. Benjamins; A. Gómez-Pérez y N. Guarino (Eds.), *Proceedings fo the ECAI'00 Workshop on Applications of Ontologies and Problem Solving Methods*, pp. 9.1–9.10. Berlin, Germany, 2000.
- STUDER, R.; BENJAMINS, V.R. y FENSEL, D.: «Knowledge Engineering: Principles and Methods». *IEEE Transactions on Data and Knowledge Engineering*, 1998, **25(1-2)**, pp. 161–197.
- STUMME, G. y MAEDCHE, A.: «FCA-MERGE: Bottom-Up Merging of Ontologies». En: N. Bernhard (Ed.), *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pp. 225–234. Morgan Kaufmann Publishers, San Francisco, California, Seattle, Washington, 2001.
- USCHOLD, M. y GRÜNINGER, M.: «Ontologies: Principles, Methods and Applications». *Knowledge Engineering Review*, 1996, **11(2)**, pp. 93–155.
- VAN HARMELEN, F.; PATEL-SCHNEIDER, P. F. y HORROCKS, I.: «Reference Description of the DAML+OIL (March 2001) Ontology Markup Language». *Informe técnico*, The DARPA Agent Markup Language Homepage, 2001. Disponible en <http://www.daml.org/2001/03/reference.html>.
- WELTY, C. y GUARINO, N.: «Supporting Ontological Analysis of Taxonomic Relationships». *Data and Knowledge Engineering*, 2001, **39(1)**, pp. 51–74.

Capítulo 6

Sistemas basados en modelos probabilísticos

Luis Daniel Hernández Molinero

Universidad de Murcia

6.1 Introducción

Los primeros sistemas expertos se basaban en un conjunto de reglas del tipo “*si (condiciones), entonces (acciones)*”. En estos sistemas de producción, o basados en reglas, se comprueba la parte *si* y, de ser cierta, se puede llevar a cabo las acciones de la parte *entonces*. Estas acciones suelen ir enfocadas a la creación de nuevo conocimiento o a la modificación del conocimiento disponible, lo que hace que estos sistemas *razonen*. Sin embargo, en muchos problemas reales las condiciones no son categóricas del tipo verdadero o falso. Una sentencia del tipo “*conducir a alta velocidad provoca accidentes*” no significa que todas las personas que conduzcan superando el límite de velocidad permitido necesariamente vuelquen o se estrellen. Sin embargo sabemos, al menos estadísticamente, que la sentencia es parcialmente cierta, relativamente cierta o que presenta excepciones. En general, el conocimiento humano está formado por afirmaciones y reglas que no siempre son totalmente ciertas.

La naturaleza de esta incertidumbre y que manifestamos en nuestras sentencias sobre el mundo que nos rodea es variada. La hay que procede de la dificultad de entender los términos lingüísticos que usamos en el habla. La afirmación “*Daniel tiene 50 años y es viejo*” no presenta incertidumbre en la edad de Daniel, sino en identificar dicha edad con la propiedad ser-viejo. Realmente se trata de una condición que no delimita claramente dos tipos de clases, ser-viejo y no-ser-viejo, sino que establece distintos *grados* de vejez. Este tipo de incertidumbre se cataloga como **incertidumbre vaga**, se soluciona mediante el establecimiento de **grados de veracidad** y es motivo de estudio de la **lógica borrosa**. El capítulo 7 estudiará cómo modelar estas situaciones con más detalle.

Otro tipo de incertidumbre es la referente a cómo de creíble son las sentencias. La afirmación “*en el partido de hoy ganará mi equipo*” tampoco se puede decir que sea

totalmente cierta; aunque lo será cuando conozcamos el resultado del partido. Ahora, la incertidumbre está asociada a lo que uno pueda creer de qué pasará a tenor de las percepciones que se estén recibiendo (condiciones del terreno, jugadores de baja, etc). Este tipo de incertidumbre se cataloga como **incertidumbre ambigua**, se soluciona estableciendo **grados de creencia** y es motivo de estudio del **razonamiento con incertidumbre**. Existen muchos tipos de medidas para establecer grados de creencia: la probabilidad, medidas acotadas, medida de Dempster-Shafer, etc. En este capítulo se estudiará cómo *modelar* y tratar el conocimiento incierto utilizando probabilidades y cómo se pueden *representar* mediante un tipo particular de modelo gráfico, las redes bayesianas.

Nuestro marco de trabajo será la **teoría de la probabilidad**, lo que garantiza mecanismos consistentes para que la información pueda ser combinada (para obtener información más general) y proyectada (para obtener información más particular). *La probabilidad permite establecer un grado de creencia (entre 0 y 1) sobre cada oración, valor que resume toda la incertidumbre derivada de nuestra ignorancia en el contexto en que ésta se afirma.* Asignar probabilidad 0 a una oración significa que creemos que la oración es falsa. Con una probabilidad 1 la creemos totalmente cierta. Lo normal es que a priori se establezcan valores intermedios.

Los **modelos gráficos** son una fusión entre la teoría de la probabilidad y la teoría de grafos que proporcionan una herramienta de representación del conocimiento incierto y complejo. Cuenta con el formalismo de los modelos teóricos probabilísticos y con la componente de la estructura gráfica, intuitiva y manipulable del usuario. Además, proporcionan un marco general que permite ver muchos modelos probabilistas como casos particulares; por ejemplo, los modelos mixtura, análisis factorial, modelos de Markov ocultos o filtros de Kalman. Se aplica a campos variados como estadística, sistemas ingenieriles, teoría de la información, reconocimiento de patrones o sistemas de apoyo a la decisión. En definitiva, proporcionan un marco general donde distintas comunidades de investigadores tienen cabida. Esto ha hecho de los modelos gráficos una herramienta cada vez más utilizada en una mayor cantidad de campos. Se podría decir que el núcleo teórico fundamental de estos sistemas se ha desarrollado en los últimos quince años del siglo XX y, aunque hoy en día se siguen obteniendo más resultados teóricos, desde principios del siglo XXI se busca más su aplicación en la resolución de problemas difíciles y complejos como el reconocimiento del habla, sistemas tutores inteligentes, predicción meteorológica, diagnóstico de fallos en maquinaria industrial o minería de datos.

De entre todos los modelos gráficos aquí sólo se hará referencia a las **redes bayesianas**. Son grafos dirigidos acíclicos que permiten representar de un modo compacto grandes modelos probabilísticos. Entre los temas a estudiar se encuentran la obtención de la mejor red que represente un determinado conocimiento, qué fuentes de información están disponibles y cómo usarlas para construir el modelo, cómo se puede razonar con estas representaciones y qué tipos de razonamientos son posibles o qué se debe hacer si se incluyen acciones (o decisiones) en la red. Cada uno de estos temas necesitaría varios libros monográficos. Aquí sólo se tratarán aspectos muy básicos en lo referente a la representación e inferencia.

El capítulo se desarrolla en dos grandes bloques. El primero, correspondiente a la sección 6.2, tiene como objetivo presentar la **teoría de la probabilidad**. Se establece un marco de referencia (sección 6.2.1) y se estudia su semántica (sección 6.2.2). Igualmente se presentan los elementos básicos que se utilizan en la representación probabilística y unas técnicas mínimas de asignación de probabilidades (apartado 6.2.3). También se muestran aspectos básicos para la construcción de *sistemas expertos probabilísticos*: por un lado, los conceptos de condicionamiento e independencia (apartado 6.2.4), pues son fundamentales para el tratamiento de redes bayesianas; y, por otro, la regla de Bayes (sección 6.2.5), que lo es para el problema de la inferencia. La segunda parte, secciones 6.3 y 6.4, se centra en el estudio de las **redes bayesianas**. Comienza introduciendo los conceptos de red causal y criterio de d-separación que es un criterio topológico de independencia de variables (apartado 6.3.1). Posteriormente (sección 6.3.2), se amplían al contexto de redes bayesianas y se explica cómo usar el criterio de d-separación para construir este tipo de redes. Por último (sección 6.4), nos introducimos en el mundo del razonamiento presentan distintos algoritmos para el razonamiento deductivo e inductivo.

6.2 Conceptos básicos de la teoría de la probabilidad

En sus orígenes, el cálculo de probabilidades se centró en intentar modelar la incertidumbre asociada a ciertos fenómenos físicos. Más concretamente, sus inicios pueden situarse en los estudios de Pierre de Fermat y Blaise Pascal (1654) sobre juegos de azar. Muy posteriormente, [Kolmogorov, 1933] desarrolló la definición de un marco teórico que se utiliza en la actualidad como el fundamento matemático para todas las aplicaciones que usan probabilidades. Existen otros marcos, pero el de Kolmogorov es el más utilizado y es el que aquí se adopta (sección 6.2.1). Esta axiomática presenta la ventaja de que, en esencia, se centra en cuál es el rango de valores que puede tomar una función de probabilidad y en una condición de aditividad. Es decir, es una axiomática que está más enfocada al **cálculo de probabilidades**. Pero detrás de estas normas de cálculo hay toda una filosofía de qué se entiende por probabilidad. Sin entrar en polémicas, se verán algunas de sus interpretaciones (sección 6.2.2).

6.2.1 Fundamentos del cálculo de probabilidades

La teoría de la probabilidad está relacionada con fenómenos en los que se cumple la **condición de azar**; esto es, aquellos en los que resulta imposible predecir sus resultados. En esta teoría cada uno de los resultados que produce uno de tales fenómenos (**o experimentos**) recibe el nombre de **resultado**. Ejemplos de este tipo de experimentos son el lanzamiento de monedas con los resultados *cara* y *cruz*; elegir a una persona en un campus universitario y relacionar su actividad con alguno de estos resultados: *alumno*, *profesor* u *otro*; seleccionar a un cliente de un banco y determinar la cantidad de euros que hay en su cuenta considerando que un resultado es cualquier cantidad de euros posible.

Un experimento estará bien definido cuando se haya establecido claramente su conjunto de resultados. Obsérvese que el conjunto posible de resultados puede ser finito o infinito. El conjunto de todos los resultados de un experimento aleatorio recibe el nombre de **espacio muestral** y cada uno de los elementos del conjunto (es decir, cada resultado) recibe el nombre de **suceso elemental** (también llamado *caso posible* o *punto muestral*). Así, el espacio muestral del lanzamiento de una moneda puede representarse mediante el conjunto $U = \{c(\text{= cara}), x(\text{= cruz})\}$ y el del lanzamiento de dos monedas sería el conjunto $U = \{(c, c), (c, x), (x, c), (x, x)\}$ donde el primer elemento de cada par ordenado representa el resultado de la primera moneda y el segundo elemento indica el caso obtenido en la segunda moneda.

En espacios muestrales finitos, un **suceso** de un experimento aleatorio es todo subconjunto del espacio muestral asociado a dicho experimento. Por ejemplo, para el lanzamiento de un dado, con espacio muestral $U = \{1, 2, 3, 4, 5, 6\}$, el suceso de que al lanzar el dado salga un número par viene dado por el conjunto $A = \{2, 4, 6\}$, o el suceso de que salga un número múltiplo de tres es el conjunto $B = \{3, 6\}$. Hay dos casos particulares de sucesos que siempre existen en todo espacio muestral:

- El suceso formado por todos los resultados posibles, $A = U$, y recibe el nombre de **suceso seguro**; y
- el formado por resultados que no se pueden presentar nunca, $A = \emptyset$, que recibe el nombre de **suceso imposible**.

Para el ejemplo del lanzamiento de dado el suceso seguro sería obtener un valor entre 1 y 6, $A = \{1, 2, 3, 4, 5, 6\}$, y un suceso imposible es obtener un valor superior a 6, $A = \emptyset$.

Con los conceptos expuestos de espacio muestral y sucesos, ya estamos en condiciones para definir qué se entiende por probabilidad:

Definición 6.1 (Probabilidad de Sucesos). *Se dice que P es una función de probabilidad, o simplemente una probabilidad, sobre el espacio muestral U , si P es una aplicación sobre sucesos que toma valores en el intervalo $[0, 1]$, verificando los siguientes axiomas:*

Axioma 1: $P(U) = 1$.

Se considera que la máxima probabilidad se establece a 1 y dicho valor se debe establecer solamente cuando se sepa que el suceso es totalmente cierto. Como siempre es cierto que va a ocurrir un resultado del espacio muestral, la probabilidad del suceso seguro necesariamente deberá ser 1.

Axioma 2: Para cualquier suceso A , $P(A) \geq 0$.

Una interpretación intuitiva de este axioma es la siguiente. Por el Axioma 1, la máxima masa de probabilidad, 1, se distribuye sobre todo el espacio muestral U . Como un suceso, A , es cualquier región del espacio muestral, se establece su masa de probabilidad proporcional al área que ocupa. Como cualquier región tiene un área no negativa, su probabilidad deberá ser positiva o nula (si la región está vacía).

Axioma 3: Para cualquiera dos sucesos A y B mutuamente excluyentes o incompatibles (matemáticamente $A \cap B = \emptyset$), entonces $P(A \cup B) = P(A) + P(B)$

A este axioma se le conoce por la **propiedad de aditividad** y lo que establece es cómo calcular la probabilidad de dos sucesos que no comparten elementos muestrales (o que producen resultados diferentes).

Al par (U, P) se le llama **espacio probabilístico** y se suele decir que P es una **función de probabilidad**¹ sobre U .

Algunos resultados inmediatos que pueden deducirse de la axiomática anterior son los siguientes:

- $P(\emptyset) = 0$.
- $P(U/A) = 1 - P(A)$.
- $\forall A, B \subseteq U, P(A \cup B) = P(A) + P(B) - P(A \cap B)$.
- Para cualquier sucesión finita de sucesos A_1, A_2, \dots, A_n mutuamente excluyentes o incompatibles (matemáticamente $A_i \cap A_j = \emptyset$), entonces

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n P(A_i)$$

En este apartado se acaban de establecer las bases de la teoría de la probabilidad matemática mediante la aplicación directa de un conjunto de axiomas. Esta manera de proceder es muy usual en las ciencias matemáticas. Un conjunto de axiomas debe de interpretarse como las reglas básicas que todos estamos dispuestos a asumir para desarrollar el resto de la teoría. Por supuesto, no todo el mundo que trabaja en el campo de la teoría de la probabilidad está totalmente conforme con este conjunto de axiomas. Por ejemplo, ¿por qué hemos de asumir que la máxima probabilidad debe ser 1? Es por ello que no le resultará difícil encontrar otros planteamientos para introducir el concepto de probabilidad. No vamos a entrar en detalle en este *campo filosófico de la probabilidad*, pero sí conviene destacar que existen distintas interpretaciones de ésta.

6.2.2 Interpretaciones de la probabilidad

Como es sabido, uno de los objetivos de las matemáticas es formalizar el significado de conceptos que usamos cotidianamente. En este sentido, la *probabilidad matemática* intenta formalizar el concepto usual de *probabilidad* y términos relacionados como *verosimilitud*, *credibilidad*,... que usamos en nuestro lenguaje natural y social (menos formalizado). En este apartado se pretende dar una somera respuesta a la pregunta: ¿qué interpretación científica debe presentar el término lingüístico *probabilidad*? Las interpretaciones más conocidas son:

¹Para ser formales deberíamos decir más bien que P es una función de probabilidad sobre los subconjuntos de U . Pero si somos más formales aún hemos de decir que una función de probabilidad se define sobre un σ -álgebra de U , un conjunto que contiene a U , a \emptyset , y es cerrado bajo las operaciones de complementario, unión e intersección infinita de conjuntos.

Interpretación frecuentista. El concepto de probabilidad se basa en el supuesto de que se dispone de un experimento aleatorio sobre un espacio de posibles resultados, llamado espacio muestral, y, además, el experimento se puede repetir de forma indefinida bajo las mismas condiciones. Suponga que al realizar el experimento se obtiene un resultado que pertenece a un cierto subconjunto A del espacio muestral. En este caso se dice que ha ocurrido el suceso A . Al repetir el experimento, bajo las mismas condiciones, pudiera ocurrir de nuevo el suceso A , o quizás no. Si repite el experimento, digamos n veces, y denota por $h(A)$ el número de veces que ha ocurrido A , se define la **frecuencia relativa** de A como $f_n(A) = h(A)/n$.

Parece lógico pensar, o al menos así opinan los frecuentistas, que si todos los ensayos (ejecuciones del experimento) son independientes unos de otros, la aparición de A debe seguir cierta regularidad, por lo que la frecuencia relativa de A debería tender a un cierto valor. Dicho valor recibe el nombre de probabilidad. Es decir, se define la probabilidad de A , $P(A)$, como [von Mises, 1957]:

$$P(A) = \lim_{m \rightarrow \infty} f_n(A)$$

Utilizar este concepto de probabilidad implica que para determinar la probabilidad de un suceso debería realizarse una gran secuencia de situaciones, todas ellas con las mismas condiciones de partida, para comprobar si se da la situación (suceso) de interés.

Existen situaciones en las que esta interpretación no parece muy lógica ya que aun suponiendo que se pudiera hacer un muestreo (infinito) repetitivo, y ya es mucho suponer, no hay garantías de que se vayan a producir las mismas condiciones en cada ensayo. Por ejemplo, cuando hablamos de la probabilidad de que nuestro equipo de fútbol preferido gane a su eterno rival es conocido que las condiciones en las que se juega cada partido de fútbol son únicas: estado físico de los jugadores, estado climático, etc.

Interpretación clásica. En esta interpretación, introducida por Leibniz y popularizada por Laplace, se considera que todos los resultados son igualmente verosímiles, en cuyo caso, la probabilidad de un suceso A viene dada por

$$P(A) = \frac{\text{casos favorables}}{\text{casos posibles}}$$

donde *casos favorables* se interpreta como el número de formas en las que aparece A y *casos posibles* como el número de formas en los que *puede* aparecer A .

Más concretamente, esta interpretación de la probabilidad aplica el **principio de indiferencia** [Keynes, 1921]. Dicho principio establece que los sucesos elementales deberían considerarse equiprobables si no hay razones para esperar o preferir unos sobre otros. Esto es, si se considera que todos los elementos de un espacio muestral son igual de verosímiles, por el principio de indiferencia, todos los elementos deberían ser equiprobables (tienen la misma probabilidad de salir).

Según este principio, si un espacio muestral consta de n resultados, la probabilidad de cada uno de ellos debería ser la **razón** $1/n$. Por ejemplo, si considera que un dado de 6 caras no está trucado y que cada una de sus caras son igual de verosímiles, todas las caras de un dado deberían tener la misma probabilidad de salir, y dicha probabilidad debe ser $1/6$.

Aunque ésta suele ser la interpretación de probabilidad que se utiliza en el contexto de juegos de azar, conviene notar que en esta definición se identifica “*igual verosimilitud*” con “*igual probabilidad*” cuando son conceptos diferentes, tal y como se verá más adelante. Esta interpretación tiene además un problema añadido y es que no especifica cómo deben asignarse probabilidades a sucesos que no sean igualmente verosímiles como en el lanzamiento de un dado trucado.

Interpretación subjetiva. En ocasiones manejamos el concepto de probabilidad en situaciones en las que no existen realmente ni aleatoriedad, ni igual verosimilitud.

Por ejemplo, en el problema de determinar el valor θ como la *proporción actual de aprobados* en algún curso de alguna titulación universitaria, ¿cómo interpretar la sentencia $P(40\% < \theta < 60\%) = 0,8$? Es decir, ¿qué significa “*la probabilidad de que θ esté entre el 40 % y el 60 % es del 0,8*”? En este problema, la proporción de aprobados, θ , es simplemente un número que puede ser conocido o no, y que se encuentra en el intervalo $(0,4, 0,6)$ o no; y no encontramos nada aleatorio en ello. Pero suponga ahora que θ denota la proporción de *aprobados para el próximo año*. Aquí parece más fácil pensar que θ es aleatorio, pues el futuro es incierto, pero ¿podemos interpretar $P(40\% < \theta < 60\%)$ en términos de una secuencia de idénticas situaciones? La situación del próximo año, en términos de aprobados, será única: es *un evento que sólo ocurrirá una vez* y no cabe hablar del número de veces de que pueda llegar a ocurrir.

La probabilidad subjetiva surgió como un modo de hablar de probabilidades cuando las interpretaciones anteriores no puedan aplicarse. La idea principal de la probabilidad subjetiva es que la **probabilidad de un suceso refleja la creencia personal en la posibilidad** (suerte, casualidad, azar, riesgo) **de que ocurra basándose en toda la información actual disponible**. En el ejemplo de la proporción de aprobados de un curso universitario, cada uno de los profesores tendría su sentimiento personal de la posibilidad de que θ esté entre el 40 % y el 60 %, estableciendo cada uno de ellos su propio riesgo. Sin duda, este tipo de sentimientos no nos sorprenderán, ya que es muy común pensar en términos de estas probabilidades personales todo el tiempo: el resultado de algún partido de fútbol, la valoración personal del riesgo de que llueva al día siguiente, la credibilidad de ganar una apuesta, etc.

Cuando un individuo acepta esta interpretación significa que la asignación de un valor de probabilidad a un suceso refleja una evaluación basada tanto en sus aspectos subjetivos como en todas las evidencias de las que dispone, evidencias que pueden estar basadas, por ejemplo, en las interpretaciones anteriores. Es decir:

- O bien se puede apoyar parcialmente en la interpretación frecuentista pues puede considerar la frecuencia relativa de las ocurrencias de un resultado o de resultados semejantes en el pasado.

- O bien se puede apoyar en la interpretación clásica pudiendo tener en cuenta el número de resultados que considera igualmente verosímiles.

6.2.3 Variables y probabilidades

El proceso expuesto consistente en establecer un espacio muestral, identificar sus sucesos elementales y, a partir de éstos, determinar las probabilidades de cualquier otro suceso no es el más usual en la práctica. Resulta más normal establecer probabilidades sobre las denominadas *variables aleatorias*. Los aspectos básicos de esta *otra forma* de proceder es el objetivo de este apartado. Formalmente una variable aleatoria se define como:

Definición 6.2 (Variable Aleatoria). *Dado un espacio probabilístico (U, P) , una variable aleatoria es cualquier función sobre U que asigna un único valor a cada resultado del espacio muestral.*

El conjunto de valores que asigna una variable aleatoria X , Ω_X , recibe el nombre de **espacio** de X . Si el espacio de X es finito o contable se dice que X es una variable aleatoria **discreta**, en cuyo caso sus valores podrán enumerarse; es decir $\Omega_X = \{x_1, x_2, \dots, x_i, \dots\}$. En otro caso, se dice que la variable X es **continua**. En este contexto se usa la expresión $X = x$ para representar al conjunto de todos los elementos muestrales de U que, a través de X , se le asigna el valor x . Es decir,

$$X = x \text{ es el conjunto } \{e \in U \mid X(e) = x\}.$$

Por ejemplo, en el lanzamiento de dos monedas con espacio muestral $U = \{(c, c), (c, x), (x, c), (x, x)\}$, se puede definir una variable aleatoria X que indique si los dos lanzamientos producen el mismo resultado o no. Es decir, X debería asignar a cada par ordenado (a, b) la etiqueta *igual_resultado* si $a = b$, y si $a \neq b$ asignar la etiqueta *resultados_diferentes*. Pero también puede definir una variable Y que calcule la suma de los valores asignados a los dos resultados. Supuesto que asigna el valor 0 al resultado c y el valor 1 al resultado x , la siguiente tabla muestra los valores que toman las variables X e Y :

Suceso elemental, e	$X(e)$	$Y(e)$
(c, c)	<i>igual_resultado</i>	0
(c, x)	<i>resultados_diferentes</i>	1
(x, c)	<i>resultados_diferentes</i>	1
(x, x)	<i>igual_resultado</i>	2

Para este ejemplo, el espacio de la variable X es el conjunto $\Omega_X = \{\text{igual_resultado}, \text{resultados_diferentes}\}$, y el de la variable Y es $\Omega_Y = \{0, 1, 2\}$ (y ambas son discretas). También es fácil comprobar que $X = \text{igual_resultado}$ es el conjunto $\{(c, c), (x, x)\}$, y que $Y = 1$ es el conjunto $\{(c, x), (x, c)\}$.

Dada una variable aleatoria discreta, X , cabe plantearse la probabilidad de que dicha variable tome un valor concreto x_i . Dicha probabilidad se establece como sigue.

Definición 6.3 (Probabilidad Discreta). *Una función $P(X)$ es una función de distribución de probabilidad para la variable aleatoria discreta X , si está definida como sigue:*

$$P(x) = P(X = x) = \begin{cases} p_i & \text{si } x = x_i \quad \text{para algún } x_i \in \Omega_X \\ 0 & \text{si } x \neq x_i \quad \forall x_i \in \Omega_X \end{cases}$$

donde p_i es la probabilidad $P(x_i) = P(X = x_i) = P(\{e \in U \mid X(e) = x_i\})$.

Es decir, la probabilidad de que se produzca un valor $x_i \in \Omega_X$ de la variable X es la probabilidad que se le asigna al suceso de todos los elementos muestrales de U que, a través de X , se le asigna el valor x_i .

Aunque las definiciones previas de variable aleatoria y (función de distribución)² de probabilidad asociada son matemáticamente elegantes y permiten el desarrollo de toda una teoría, no muestran explícitamente cómo se pueden llegar a aplicar en la práctica. Clarifiquemos esta cuestión.

En la práctica, el conocimiento se puede presentar como un conjunto de características para las que no siempre se sabe cuáles son los valores exactos que presentan, aunque sí un riesgo relativo de que dichos valores se produzcan. Tal es el caso, por ejemplo, de las enfermedades y síntomas en un contexto médico donde se puede desconocer la enfermedad exacta de un paciente pero sí puede establecerse, con un cierto riesgo, qué le aqueja en función de los síntomas observados. En estos dominios, puede identificar cada característica con una variable (aleatoria), variable que tiene asignada una *cantidad* que pueden medirse, mediante algún mecanismo a determinar, y que para nosotros puede ser una *cantidad desconocida*. Así, todos los posibles valores de la variable corresponden a todas esas posibles *cantidades* que pueden medirse o, si lo prefiere, se corresponden con todos los diferentes estados de la característica que representa.

Cuando identifique una característica y le asocie una variable, debe preguntarse qué tipo de valor debería adoptar y cuántos valores puede tomar, pues le determinará algunas tareas del modelado. En este sentido, las variables pueden ser tanto discretas (toman un número contable de valores) como continuas (tiene una cantidad infinita de valores) y no será extraño que tenga que trabajar simultáneamente con ambos tipos de variables. En cuanto a los tipos de valores, estos pueden ser numéricos (valores enteros o reales) o nominales (valores de un conjunto, usualmente finito, de *etiquetas* establecidas). En particular, si trabajara con variables discretas³ puede afinar aún más los tipos de valores. Los más destacados son:

²Cuando se trabaja con variables aleatorias se utiliza constantemente el término *función de distribución*. Por ejemplo, se acaba de definir la *función de distribución* de probabilidad y más adelante se definirán la *función de distribución* de probabilidad conjunta o la *función de distribución* de probabilidad marginal. Para hacer más fácil la lectura, en lo que sigue, se usará solamente la palabra **probabilidad** para hacer referencia a la expresión **función de distribución de probabilidad**.

³En lo que sigue se tratarán sólo variables discretas. La razón principal para adoptar esta postura es porque se facilita la comprensión de los desarrollos. Tenga en cuenta que estos resultados se extienden al caso de variables continuas.

- Valores numéricos. Representan a valores contables como los números naturales.
- Valores lógicos o booleanos. Representan a proposiciones, a afirmaciones del mundo que se desea modelar. Como en la lógica clásica toman los valores *verdad* y *falso*.
- Valores ordenados. Se utilizan, en general, cuando ciertos valores pueden ordenarse respecto al atributo que representa la variable. Por ejemplo, la concentración de oxígeno en un metro cúbico de agua podría tomar los valores *{bajo, normal, alto}*. Estos valores suelen aparecer como resultado de **discretizar** una variable real continua.
- Valores sin orden. Son todos aquellos valores prefijados que puede tomar una atributo que no tiene por qué considerarse ordenado, actúan simplemente como etiquetas. Por ejemplo, si una calle tiene 100 viviendas, una variable que represente a los números de policía tomará todos los valores posibles desde el 1 hasta el 100.

Se acaba de introducir un concepto que puede ser nuevo para usted: la discretización. ¿Qué significa? En líneas muy generales es la conversión de un valor numérico continuo en uno discreto; más concretamente en un conjunto de valores ordenados. Cada valor ordenado representa a un intervalo numérico. Un ejemplo sencillo es la discretización de la calificación de 0 a 10 de los estudiantes, que da lugar a los valores ordenados *{suspenso, aprobado, notable, sobresaliente, matrícula de honor}* que corresponde, respectivamente, a los intervalos *{[0,4.99), [5.00,6.99), [7.00,8.49), [8.50,10.00), 10.00]}*. Notar que la discretización mantiene el orden que viene dado por los valores numéricos, pero es posible (por exigencias de su modelo) que dicho orden deba olvidarse con lo que trabajaría con valores sin orden.

Note que una vez haya identificado las variables *discretas* de interés y su conjunto de valores, éstos deberán ser valores mutuamente excluyentes y exhaustivos. Esto significa que para cada variable se contemplan todos sus posibles valores y que puede tomar exactamente sólo uno de sus valores en cada instante.

Establecidas las variables discretas, ahora deberá determinar un riesgo (probabilidad) de que se produzcan sus valores, pero ¿dónde aparece ese riesgo en el modelo? Uno podría cubrirse las espaldas diciendo que simplemente deberá considerar cada una de sus variables X , su conjunto discreto de valores $\Omega_X = \{x_i | i \in N\}$ y establecer un valor p_i para cada valor x_i como la probabilidad $P(x_i) = P(X = x_i)$; es decir, si establece que $P(x_i) = p_i$ debe garantizar que $\sum_i p_i = 1$. Pero la dificultad real está en ¿cómo se pueden establecer esos valores p_i ? Aquí entra en juego las distintas interpretaciones de la probabilidad.

A continuación, se indican algunas técnicas sencillas con un ejemplo simple. Suponga que debe establecer las probabilidades de X con $\Omega_X = \{varón, mujer\}$:

- Si supone que los valores de la variable son equiprobables, podría establecer que

$$P(varón) = P(mujer) = 1/2.$$

- Pero si dispone de muestras estadísticas, puede contabilizar su frecuencia relativa e identificar ésta con su probabilidad. Por ejemplo, si sobre una población de 1000 individuos usted observa que 600 son mujeres y el resto varones, entonces establecería que:

$$P(\text{varón}) = 4/10 = 0'4, \quad P(\text{mujer}) = 6/10 = 0'6.$$

En ocasiones, en vez de estimar las probabilidades por la proporción de varones y mujeres es interesante utilizar la **ley de sucesión de Laplace**, que establece:

$$P_{\text{Laplace}}(X = x) = \frac{\text{Casos favorables de } x + 1}{\text{Casos totales} + \text{Número de casos de } X}.$$

Para las proporciones anteriores se determinaría que:

$$P_{\text{Laplace}}(\text{varón}) = 401/1002 \approx 0'4002, \quad P_{\text{Laplace}}(\text{mujer}) = 6001/1002 \approx 0'5998.$$

La ley de sucesión de Laplace suele utilizarse cuando se tienen muestras pequeñas y de alta dimensionalidad (muchas variables y pocos casos).

- También puede asignar, para cada caso x_i de la variable, un valor p_i que refleje su grado de creencia en cada valor x_i que ocurra. Por ejemplo, si se dirige a un pueblo de 1000 personas y sin ningún tipo de observación sobre la proporción de mujeres y hombres cree que hay el triple de mujeres que de varones establecería que $3P(\text{varón}) = P(\text{mujer})$; pero como ocurre que debe cumplirse que $P(\text{varón}) + P(\text{mujer}) = P(U) = 1$ se deduce que $4P(\text{varón}) = 1$ y por tanto:

$$P(\text{varón}) = 1/4, \quad P(\text{mujer}) = 3/4.$$

Desde la perspectiva subjetiva existen distintas técnicas para obtener una probabilidad. La comentada se basa en comparar un suceso A con otro suceso. En su versión más sencilla se compara A con su complementario $A^c = U - A$ para, a continuación, establecer cuánto cree (o siente) que A puede ocurrir más veces que A^c . Al establecerse que $P(U) = 1$ entonces el cálculo de $P(A)$ es inmediato. El mismo proceso se puede hacer para establecer la probabilidad de distintos valores de una variable acotada a partir de las creencias establecidas por el usuario (Algoritmo 6.1). El proceso es sencillo. En primer lugar se establecen los casos de la variable que se consideran más y menos creíbles que los demás. De este modo ya se dispone de unas “probabilidades” iniciales. En segundo lugar, el usuario compara cada caso x_i con aquellos otros casos x_k para los que el usuario es capaz de decir cuántas veces x_i es más creíble. Cada x_i podrá ser comparado con uno, varios, o todos los demás x_k . En tercer lugar, una vez asignadas las “probabilidades” sobre las x_i , se debe comprobar si son coherentes y, de no serlo, debería repetirse el proceso anterior con más cuidado. Por último, los valores asignados se normalizan.

En esta versión simple se compara un suceso con otro suceso de su mismo espacio muestral, pero también se podría comparar entre sucesos de distintos espacios.

Algoritmo 6.1 Determinación de probabilidades subjetivas mediante creencias relativas.

Entradas: El espacio de valores, Ω_X , de una variable X .

Resultado: Una probabilidad subjetiva, $P(X)$.

- 1: Seleccionar los valores $x_m \in \Omega_X$ que se consideren menos creíbles y asignarles el mínimo valor de creencia. Es decir:

$$x_m = \arg \min_x f(x)$$

Seleccionar los $x_M \in \Omega_X$ que se consideren más creíbles y asignarles el máximo valor de creencia. Es decir:

$$x_M = \arg \max_x f(x)$$

- 2: Comparar todos los posibles valores $x_i \in \Omega_X$ con sus comparables, $x_k \in \Omega_X$, en términos de su credibilidad. Es decir, establecer relaciones del tipo $f(x_i) = c_{ik}f(x_k)$ con c_{ik} un valor numérico real. Cada igualdad debe leerse como “ x_i es c_{ik} -veces más creíble que x_k ”.

- 3: Comprobar la consistencia con que se van asignando los valores.

Es decir, seleccionar casos x_1 y x_2 a los que ya se les ha asignado valores $f(x_1)$ y $f(x_2)$ y comprobar si tienen la misma creencia relativa que reflejan. Por ejemplo, si $f(x_1) = f(x_2)$ preguntarse si esto es lo que esperaba o si por el contrario debe volver a asignar nuevas verosimilitudes.

En otros casos puede necesitar un tercer caso x_3 . Por ejemplo, si $f(x_3) = (f(x_1) + f(x_2))/2$ preguntarse si esto es lo que esperaba.

Si detecta inconsistencias, volver al paso anterior.

- 4: Establecidas todas las creencias definir:

$$P(x_i) = \frac{f(x_i)}{\sum_x f(x)}$$

En este caso se dice que se aplica la técnica de **medición directa**, que es análoga a pesar el *objeto A* en el platillo de una balanza y colocar en el otro platillo algún peso conocido. Por ejemplo, para obtener $P(A)$ se puede realizar el siguiente razonamiento.

- Considero que A es menos probable que obtener una cara en el lanzamiento de una moneda, de donde $P(A) < 1/2$.
- Considero que A es menos probable que obtener 3 caras en 3 lanzamientos sucesivos de una moneda, pero más probable que obtener 4 caras en 4 lanzamientos sucesivos de la misma moneda, de donde $1/2^4 < P(A) < 1/2^3$.
- Continuar de este modo para ir refinando nuestra creencia en A hasta llegar a la precisión deseada para calcular $P(A)$.

Una última técnica, más práctica que la anterior, para asignar la probabilidad (subjetiva) de un suceso A es la siguiente.

- Al usuario se le da a elegir dos juegos:

Juego 1. Si ocurre A , el usuario recibe un premio económico de P euros.

Juego 2. Sacar una bola de una urna con n bolas blancas y $100 - n$ bolas negras. Si la bola extraída es blanca, el usuario recibe P euros.

Es claro que si todas las bolas fuesen blancas el usuario preferiría el juego 2; pero si todas fuesen negras preferiría el juego 1. El número n para el cual los dos juegos son indiferentes (o igual de atractivos) estima la probabilidad de A por la cantidad $n/100$.

Al igual que hablamos de la probabilidad de cierta variable, podría también plantearse el establecer una probabilidad sobre dos variables X e Y . Esto es, ¿cuál sería el riesgo de que la variable X tome un valor x y una variable Y tome un valor y simultáneamente? Dicho riesgo se denota por $P(x, y) = P(X = x, Y = y)$ y al conjunto de todos los valores probabilísticos $P(x, y)$ recibe el nombre de **(función de distribución de) probabilidad conjunta** de X e Y . Este concepto puede generalizarse a tres o más variables:

Definición 6.4 (Probabilidad Conjunta). *Sean n -variables $X = \{X_1, X_2, \dots, X_n\}$, todas ellas discretas. Una función P que asigna un número real $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$ a cada combinación de valores $x = (x_1, x_2, \dots, x_n)$ recibe el nombre de probabilidad conjunta de las variables aleatorias X si verifica las siguientes condiciones:*

- Para cada combinación de valores $x = (x_1, x_2, \dots, x_n)$

$$0 \leq P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \leq 1$$

- La suma de todos los números reales es 1:

$$\sum_{x_1, x_2, \dots, x_n} P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = 1$$

En lo que sigue indicaremos por $P(x_1, x_2, \dots, x_n)$ al valor

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n),$$

y por $P(X_1, X_2, \dots, X_n)$ al conjunto de todos estos valores que suman la unidad.

Dada una probabilidad conjunta $P(X, Y)$ sobre dos variables, puede construirse una probabilidad sobre X asignando valores según la expresión:

$$P(x) = P(X = x) = \sum_y P(x, y) = \sum_y P(X = x, Y = y) \quad (6.1)$$

Dicha probabilidad recibe el nombre de **probabilidad marginal** de X . Este concepto puede también extenderse a dos o más variables. Así, dada una probabilidad conjunta $P(X, Y, Z)$ cabe plantearse determinar la probabilidad marginal sobre una sola variable $P(x) = \sum_{y,z} P(x, y, z)$ o sobre dos variables $P(y, z) = \sum_x P(x, y, z)$. En general,

Definición 6.5 (Probabilidad Marginal). *Sea una probabilidad conjunta, P , sobre el conjunto de variables $X = \{X_1, X_2, \dots, X_n\}$ y considere el subconjunto $Y \subset X$ formado por las i primeras variables de X . Se define la probabilidad marginal de Y como la función con valores*

$$P(x_1, x_2, \dots, x_i) = \sum_{x_{i+1}, x_{i+2}, \dots, x_n} P(x_1, x_2, \dots, x_i, x_{i+1}, \dots, x_n) \quad (6.2)$$

Observe que no existe ningún tipo de restricción en considerar sólo las i primeras variables: sólo tendrá que reordenar el conjunto para poner *al principio* las variables en las que está interesado. Calcular la probabilidad marginal sobre las variables $\{X_1, X_2, \dots, X_i\}$ se puede leer como *eliminar por sumas las variables $\{X_{i+1}, \dots, X_n\}$ de la probabilidad conjunta sobre X* donde **eliminar por sumas una variable X_i** se interpreta como sumar los valores de la distribución de probabilidad en todos los valores x_i de esa variable. Esta idea de *eliminación* será fundamental en los procesos de inferencia y se insistirá en ella más adelante.

En lo que sigue, al hablar de una variable X no se hará distinción entre si dicha variable es sólo una variable del modelo o si es está formada por un conjunto de variables individuales. Por ende, al hablar de su probabilidad $P(X)$ no se especificará si es conjunta o no. Sus significados se inferirán del contexto.

6.2.4 Probabilidad condicionada e independencia

Los siguientes conceptos que vamos a desarrollar en este apartado son de los más importantes en el cálculo de probabilidades y constituyen la base de la construcción de los modelos gráficos probabilísticos.

Definición 6.6 (Probabilidad Condicionada). *Dadas dos variables X e Y se define la probabilidad condicionada de X , dado un valor concreto y' de Y como el conjunto de valores $P(x|y')$ dados por la expresión:*

$$P(x|y') = P(X = x|Y = y') = \frac{P(x, y')}{P(y')} \text{ siempre que } P(y') \neq 0 \quad (6.3)$$

En el caso de que $P(y') = 0$, entonces la probabilidad condicionada está sin definir. Tenga presente que en una expresión $P(x|y)$, y es un valor fijo de Y y que x es un valor que se mueve en todo el dominio de X . Al conjunto de todos los valores $P(x|y)$, para un y fijo, se suele notar por $P(X|y)$, que es una probabilidad sobre X , y se representa por $P(X|Y)$ al conjunto formado por todas las probabilidades $P(X|y)$ para cada valor y de Y .

Intuitivamente, una probabilidad condicionada sobre un conjunto de variables, X , no es más que una probabilidad sobre dichas variables pero restringida al subespacio definido por valores establecidos, y , de otras variables, Y (véase el Ejercicio 6.4).

Es curioso notar que desde el punto de vista subjetivo, **todas las probabilidades subjetivas son condicionadas**, ya que si $P(X)$ representa los grados de creencia personal en los valores de X basándose en **toda** la información actual disponible, para

ser explícitos dicha probabilidad debería expresarse como $P(X|I)$ donde I representa toda la información que se conoce. Pero si todas las probabilidades subjetivas son asignaciones subjetivas y propias del individuo ¿para qué sirve entonces la Ecuación (6.3)? La respuesta pasa por exigir al investigador que sea muy cuidadoso cuando establezca probabilidades: la asignación subjetiva de pesos debe de ser coherente con la información ya disponible. Por ejemplo, la asignación de valores para $P(x|y)$ debería ser coherente con la asignación realizada con $P(x, y)$ y $P(y)$. El uso de fórmulas y teoremas en la perspectiva bayesiana queda justificado para garantizar el comportamiento probabilístico del investigador y facilitar las tareas de asignación. La expresión (6.3) y otros resultados, como el teorema de la probabilidad total o la regla de Bayes, juegan este importante papel.

Una cuestión interesante, y que surge de forma natural a partir del condicionamiento, es saber si el que se produzcan ciertos casos (fijos) de una variable, puede determinar o influir en los casos que pueden tomar otras variables o, si por el contrario, son totalmente independientes de cualquier valor que se fije. Este concepto de dependencia e independencia entre variables es el que se formaliza a continuación.

Definición 6.7 (Independencia de Variables). *Sean X , Y y K tres conjuntos de variables disjuntas.*

- Si $P(X|Y) = P(X)$, se dice que las variables X e Y son (*marginalmente*) independientes y se notará por $I(X, Y)$.

Es decir, para cualesquiera valores x e y , se cumple que $P(x|y) = P(x)$. Los valores y no influyen en la creencia que se tiene sobre la ocurrencia de los valores x , y viceversa.

- Si $P(X|K, Y) = P(X|K)$, se dice que las variables son **condicionalmente independientes** dado el conocimiento K y se notará por $I(X, Y|K)$.

Intuitivamente significa que en el contexto del conocimiento adquirido u observado K , la creencia o probabilidad que se tenga en X no se ve afectada por la información que pueda suministrar los valores de Y . Notar que la independencia marginal es un caso particular de la condicional: $I(X, Y|\emptyset) = I(X, Y)$.

Cuando la independencia no se cumple se dice que las variables X e Y son marginalmente dependientes o condicionalmente dependientes dado K , según la igualdad violada.

El concepto de dependencia está relacionado con el de **correlación**: $P(x|y) > P(x)$ indica que la presencia de y favorece la presencia del valor x y se habla de **correlación positiva** entre ambos valores; si la presencia de y desfavorece la presencia de x , $P(x|y) < P(x)$, se habla entonces de **correlación negativa**. Cuando dos variables, X e Y , son ordinales (sus valores están ordenados) también puede hablarse de correlación entre variables. Más concretamente, si cuanto mayor sean los valores de Y más probable es que aumenten los valores de X y cuanto menor sean los valores de Y más probable es que disminuyan los valores de X , se habla entonces de correlación positiva entre las variables X e Y . Por el contrario, existe una correlación negativa

entre las variables cuando la correlación entre todos sus valores es negativa: cuanto mayor sean los valores de Y más disminuyen los valores de X , y viceversa.

Un ejemplo de correlación positiva es la relación entre las variables precio y peso de un producto: cuanto más pague más kilos del producto tendrá y cuanto menos kilos adquiera menos tendrá que pagar. La relación entre la edad y la capacidad de memoria a corto plazo están, por contra, correlacionadas negativamente: cuanto más viejos nos hacemos más probable es que olvidemos lo que hicimos el día anterior.

6.2.5 Regla de Bayes

En este apartado se muestran los resultados (o teoremas) más importantes del cálculo de probabilidades y que constituyen la base de los motores de inferencia de los sistemas inteligentes probabilísticos. El más importante es la regla de Bayes que, en su versión más simple, establece que:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad (6.4)$$

Se puede interpretar (6.4) como sigue. Nuestro modelo parte de una probabilidad ‘a priori’ $P(X)$ sobre la credibilidad de los valores x de X . Entonces, en algún instante, observamos que otra variable Y se instancia al valor y . Ya sabe que cómo de probable es un valor x de X cuando Y se ha instanciado al valor y viene dado por la probabilidad (condicionada) $P(x|y)$. Usted ya sabe cómo obtenerla a partir de $P(x, y)$ pero ¿cómo obtenerla a partir de aquella probabilidad inicial $P(X)$? La regla de Bayes nos da la solución y puede entenderse como **una fórmula para pasar de la probabilidad ‘a priori’, $P(X)$, a una probabilidad ‘a posteriori’, $P(X|y)$.** La modificación consiste en multiplicar por la razón $\frac{P(y|x)}{P(y)}$. Es decir, la regla de Bayes describe cómo cambia la probabilidad cuando aprendemos nueva información.

Es usual interpretar cada valor x de X como una posible hipótesis en la resolución de un problema y cada uno de los valores y como las observaciones, evidencias o hallazgos que se van detectando en el tiempo y que refuerzan unas hipótesis sobre otras. Esto es, ante la observación de una evidencia y_1 la regla de Bayes cambia la probabilidad de la hipótesis $P(x)$ a una probabilidad actualizada $P(x|y_1)$ en la que se contempla esa nuevo dato y_1 . Si se diera un nuevo hallazgo y_2 , la regla de Bayes permite pasar a una nueva probabilidad modificada $P(x|y_1, y_2)$ a partir de $P(x|y_1)$ para contemplar ese último dato y_2 ; y así suscesivamente. Un ejemplo clásico es el campo médico donde X representa una enfermedad que puede manifestarse mediante una serie síntomas $\{Y_1, Y_2, \dots, Y_n\}$. Este modelo, gráficamente, se puede contemplar como se muestra en la Figura 6.1. El médico ante la presencia positiva o negativa de dichos síntomas establece una diagnóstico sobre la presencia de la enfermedad X . El diagnóstico no se suele establecer en términos de total certidumbre, sino que estaría sometido a un cierto riesgo subjetivo del propio médico pero que, matemáticamente, se puede determinar mediante la regla de Bayes.

Es interesante y esclarecedor estudiar más detenidamente cada uno de los factores que aparecen en (6.4). A saber, una probabilidad a priori, una versosimilitud y una constante de normalización.

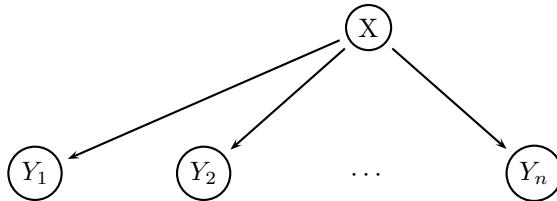


Figura 6.1: Conexión divergente. Puede representar la relación entre una causa, X , y los efectos que produce, Y_i .

Probabilidad a priori. La probabilidad $P(x)$ es conocida como **probabilidad a priori** y representa el conocimiento incierto del usuario sobre la ocurrencia del caso x de X . Esta probabilidad se obtiene a partir de toda la información disponible y existen distintas técnicas para su obtención. Buena parte del apartado 6.2.3 se dedicó a este cometido, mostrando algunas técnicas básicas para calcularlas.

Verosimilitud. La probabilidad $P(y|x)$ es conocida como **verosimilitud**. Más concretamente, $P(y|x)$ es la verosimilitud o creencia de x dada por y . Recuerde que y es un valor fijo y que x toma valores en su dominio, por lo que $P(y|x)$ no se interpreta como una probabilidad condicionada. Es por ello que algunos autores prefieren denotar dicha función por $L(x|y)$.

La función de verosimilitud $L(x|y) = P(y|x)$ hay que considerarla como una **función de x para un y fijo**, pero como además $P(y|x)$ representa el grado de creencia del investigador de que los datos Y tomen el valor y dada la **información hipotética** de que X tome otros determinados valores x , los **valores $L(x|y) = P(y|x)$ se deben obtener a partir de las probabilidades condicionadas de y para los distintos valores de x** . Así, dicha función deberá ser tal que un x para el cual $P(y|x)$ toma un valor alto significa que:

- por un lado, ese x es más probable de que sea cierto frente a otro caso x' para el que su $P(y|x')$ es más pequeño,
- por otro lado, que y debería ser más plausible en su ocurrencia cuanto mayor sea el valor $P(y|x)$.

En términos de hipótesis/evidencias una función de verosimilitud debería establecer que las hipótesis x que tengan una mayor verosimilitud por y deberían, de algún modo, tener una probabilidad mayor cuando se ha observado que y ocurre. Traduzcamos todo lo anterior a un ejemplo. Si se asoma a la ventana de su casa y observa un artilugio de metal con cuatro ruedas ¿por qué piensa que es un coche?, sencillamente porque esos elementos aparecen en todos los coches. Podría tener alguna duda de si se trata de una moto pero seguro que no piensa que sea una persona. En notación formal, si el conocimiento se modela del siguiente modo:

- y es el dato que establece de que hay presencia de metal y cuatro ruedas.
- X es una variable (hipótesis) con valores

$$\Omega_X = \{x_1 = \text{coche}, x_2 = \text{moto}, x_3 = \text{persona}\}$$

Entonces, se pueden establecer las siguientes verosimilitudes:

- La afirmación “los coches generalmente se parecen a lo que he visto” es probablemente muy cierta, lo que implica que $P(y|x_1) \approx 1$.
- La sentencia “las personas generalmente se parecen a lo que he visto” se considera muy poco creíble, lo que implica que $P(y|x_3) \approx 0$.
- La oración “las motos generalmente se parecen a lo que he visto”, implica que $P(y|x_2)$ estará más próximo de $P(y|x_1)$ que de $P(y|x_3)$.

Note que $P(y|x_1)$ nos da dos informaciones. Por un lado, que los coches son la hipótesis más creíble y que los datos observados son los más plausibles si fuese un coche lo que estamos observando.

Constante de normalización. Retomando la expresión (6.4) nos queda por comentar el término $P(y)$. Dicho valor es una constante ya que y es un valor fijo en dicha expresión. Para obtener dicho valor tiene dos alternativas:

- Considerar la probabilidad conjunta $P(x, y)$ y calcular el valor marginal de $P(y)$ aplicando la Ecuación (6.1).
- O aplicar el **teorema de la probabilidad total**. Dadas dos variables X e Y este resultado establece que

$$P(y) = \sum_x P(y|x)P(x) \quad (6.5)$$

Sustituyendo (6.5) en la regla de Bayes se tiene

$$P(x|y) = \frac{P(y|x)P(x)}{\sum_x P(y|x)P(x)} \quad (6.6)$$

La Ecuación (6.6) es muy interesante porque nos dice que se puede calcular una probabilidad a posteriori usando solamente una probabilidad a priori y una función de verosimilitud. Es decir, una vez que conozca todos los numeradores de la regla de Bayes para calcular $P(X|y)$, sólo tiene que eliminar por sumas la variable X para obtener el denominador.

Al ser $P(y)$ una constante, la regla de Bayes puede también expresarse como:

$$P(x|y) = \alpha P(y|x)P(x) \quad P(x|y) \propto P(y|x)P(x)$$

donde α es dicha constante de normalización que se calcula por el teorema de la probabilidad total.

Un caso particular de la regla de Bayes es cuando la probabilidad a priori de las distintas hipótesis (causas o escenarios), $P(x)$, son la misma, lo que conduce a:

$$P(x|y) = kP(y|x)$$

con k una constante que es independiente de cada x . Es decir, si no tenemos razones para preferir unas hipótesis frente a otras sus probabilidades son proporcionales a sus verosimilitudes (véase el Ejercicio 6.1).

6.3 Una introducción a las redes bayesianas

En la sección 6.2 se le ha presentado el fundamento del cálculo de probabilidades. A partir de ahora se utilizarán todos estos fundamentos para introducirle en el punto clave que permite construir toda una tecnología computacional basada en estructuras gráficas para la representación y el razonamiento probabilístico en inteligencia artificial: las redes bayesianas. Una de las grandes ventajas de esta representación del conocimiento incierto es que no sólo permite construir técnicas de razonamiento basadas en el cálculo probabilístico, sino que, además, codifica en la red toda una semántica sobre las independencias subyacentes en las variables que pueden ser detectada mediante observación de la topología (a golpe de vista) y sin necesidad de cálculo alguno.

Antes de continuar conviene revisar la terminología de la teoría de grafos que se usará en el resto del capítulo. Un grafo G es una pareja (V, R) , donde V es un conjunto finito, no vacío, de elementos llamados **nodos** (o vértices); y R es un conjunto de pares ordenados con términos en V . Si $(X, Y) \in R$ se dice que hay un **arco** de X a Y , que X es **padre** de Y y que Y es **hijo** de X . Gráficamente se indica por $X \rightarrow Y$. Si además R es una relación simétrica, contiene a las parejas (X, Y) e (Y, X) , entonces se dice que R está formado por **aristas**. Gráficamente una arista entre dos nodos se representa por $X - Y$. Se dice que hay un **enlace** entre dos nodos si existe un arco o una arista entre ambos, en cuyo caso se dice que están relacionados, son vecinos o son adyacentes. Si un conjunto de nodos $\{X_1, X_2, \dots, X_n\}$, con $n \geq 2$, es tal que $(X_{i-1}, X_i) \in R$ para $2 \leq i \leq n$, entonces el conjunto de enlaces que relacionan a los nodos del conjunto recibe el nombre de **camino**. Cuando los enlaces son aristas también recibe el nombre de camino no dirigido; pero si los enlaces son arcos se suele hablar de **camino dirigido**. Si existe un camino dirigido entre un nodo X y un nodo Y se dice que X es un **ancestro** de Y y que Y es un **descendiente** de X . Y es un **no-descendiente** de X si Y no es un descendiente de X . Una ordenación de todos los nodos de V se dice que es una **ordenación ancestral** si se cumple que si X es un descendiente de Y , entonces X está después de Y en el orden. Un **árbol** es un grafo dirigido donde cada nodo tiene un único parente, excepto el nodo raíz que no tiene ningún parente. Un **poliárbol** es un grafo simplemente conectado, es decir, grafos dirigidos tales que el grafo no dirigido subyacente no presenta ciclos.

6.3.1 Redes causales y d-separación

La Figura 6.1 visualiza gráficamente la situación médica en la que X representa a una enfermedad que puede manifestarse a través de los síntomas $\{Y_1, Y_2, \dots, Y_n\}$. Pero la figura también puede utilizarse para representar gráficamente cualquier otra relación del tipo “una causa produce muchos efectos” en la que cada flecha (o arco) del tipo $X \rightarrow Y_i$ define una dependencia causal del nodo parente X (la causa) al nodo hijo Y_i (el efecto). Es decir, el estado del parente tiene un impacto en (léase, es responsable de) el estado de su hijo.

Así, un modo de representar una situación causal (bajo incertidumbre) consiste en partir de un conjunto de variables \mathcal{V} y para cada par de variables $X, Y \in \mathcal{V}$ establecer

un arco de X a Y si y sólo si X es una causa de Y siempre que el grafo resultante sea acíclico (véase el Ejercicio 6.7). En el contexto de la representación del conocimiento el grafo resultante se conoce como **red causal**. La condición de ser acíclico significa que no existe ninguna variable en el grafo tal que si seguimos la dirección de las flechas volvamos a él.

Una red causal, como en cualquier otro grafo, se compone de nodos y arcos. Cada nodo es una variable y representa el conjunto de posibles estados de un evento. Cada variable toma valor en exactamente uno de sus estados que, para nosotros, puede ser totalmente desconocido. Cuando se conoce con total certeza el valor que toma una variable se dice que la variable está **instanciada** y que dicho valor es una **evidencia** o **instancia** sobre la variable. Los arcos entre las variables de la red definen una **topología**; es decir, el modo en que las variables se conectan. La ventaja de esta representación gráfica del conocimiento es que puede identificar grupos de variables con estructuras especiales, o *patrones topológicos*, y cada estructura le permite establecer cómo la evidencia sobre una variable puede afectar a la certidumbre sobre las demás variables del grafo. Así, por *patrón topológico* se debe entender a una regla que establece un tipo concreto de razonamiento. Se identifican tres patrones básicos [Jensen, 1996, 2001]: conexión en serie, conexión divergente y conexión convergente.

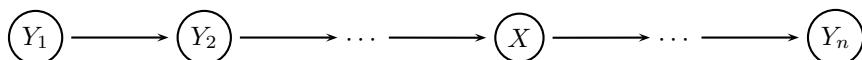


Figura 6.2: Conexión en serie. Cuando una variable es causa y efecto a la vez.

Conexión en serie

Una conexión en serie responde a la Figura 6.2. Y_1 condiciona los resultados de Y_2 , la cual es causa de Y_3 , y así sucesivamente hasta Y_n . Cuando se sabe con certeza el valor de Y_1 , éste influirá sobre la certidumbre de Y_n por ese encadenamiento en serie de las variables. Análogamente una evidencia sobre Y_n transmite certidumbre hasta Y_1 . Sin embargo, si una variable intermedia X es conocida (está instanciada), las variables Y_1 e Y_n dejan de influirse mutuamente. En efecto, si inicialmente disponemos de una evidencia sobre X , ésta transmitirá cierta información hasta Y_n ; pero una posterior instancia de Y_1 no modificará en absoluto nuestra credibilidad en los valores de X (puesto que ya lo conocíamos antes de saber nada sobre Y_1) y por tanto la instancia de Y_1 no cambiará nuestra certidumbre sobre Y_n . Es decir, la credibilidad sobre los valores de Y_n viene dada por la primera evidencia sobre X y no por el posterior conocimiento sobre Y_1 . De aquí la siguiente regla:

Regla 1: La evidencia se transmite de un extremo a otro de una conexión en serie, salvo que una variable intermedia esté instanciada.

Es decir, las variables Y_1 e Y_n son dependientes cuando no sabemos nada sobre las variables intermedias. Pero cuando una de tales variables intermedias es conocida

(léase X) entonces se bloquea el flujo de información e Y_1 e Y_n pasan a ser independientes. Se dice entonces que Y_1 e Y_n están **d-separadas** dada X , o que X bloquea a Y_1 e Y_n .

Conección divergente

Responde a la ya comentada Figura 6.1. En este caso, la información de cada una de las variables Y_i pasa a cualquier otra variable Y_j a través de X . Una evidencia sobre Y_1 modificará nuestra certidumbre en X y ésta nuestra credibilidad en Y_2 , Y_3 , ..., Y_n . Sin embargo, si previamente se hubiese recibido evidencia sobre X , será ésta y no la evidencia de Y_1 , la que establece la credibilidad del resto de las variables Y s. En esta situación la regla obtenida es:

Regla 2: La evidencia se transmite entre los hijos de una conexión divergente, salvo que la variable padre esté instanciada.

Es decir, dos variables Y_i e Y_j son dependientes cuando no sabemos nada sobre la variable intermedia X . Pero cuando X es conocida entonces se bloquea el flujo de información e Y_i e Y_j pasan a ser independientes. Se dice entonces que Y_i e Y_j están **d-separadas** dada X .

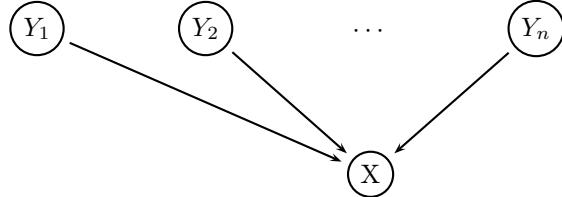


Figura 6.3: Conexión convergente. Puede representar la relación entre las distintas causas, Y_i , que producen un mismo efecto X .

Conección convergente

Corresponde a la situación de la Figura 6.3. Cuando un evento o efecto X es desconocido, una de las causas que puede producirlo no nos informa de nada sobre el resto de las causas que pudieran provocarlo. Esto es, las variables Y_i son independientes. Pero si X es conocido, la evidencia de una causa produce un cambio de credibilidad en que el resto de las causas hayan podido producir el efecto. De donde se obtiene la regla:

Regla 3: La evidencia no se transmite entre los padres de una conexión convergente, salvo que la variable que las conecta esté instanciada o uno de sus descendientes sea conocido.

Así pues, las tres reglas anteriores indican cuándo la evidencia puede o no propagarse entre los nodos. Estas tres reglas que establecen cuándo, topológicamente, dos variables son independientes dada una tercera son conocidas como criterio de separación de independencias, o **criterio de d-separación** [Pearl, 1988].

Definición 6.8 (Criterio de d-separación). *Dos variables X e Y están d-separadas si para todos los caminos entre X e Y hay una variable intermedia Z (distinta de X e Y) cumpliéndose alguna de estas dos condiciones:*

- *O siguen un patrón en serie o un patrón divergente y Z está instanciada.*
- *O siguen un patrón convergente y ni Z ni ninguno de sus descendientes están instanciados.*

Existen otros criterios equivalentes al de d-separación. Una alternativa es la que se muestra en el Algoritmo 6.2, que se debe a Lauritzen cia. (1990).

Algoritmo 6.2 Comprobar si dos conjuntos de variables están d-separados por un tercero.

Entradas: Tres conjuntos disjuntos de variables \mathcal{X} , \mathcal{Y} y \mathcal{Z} .

Resultado: Saber cuándo el conjunto \mathcal{X} está d-separado de \mathcal{Y} por el conocimiento de un tercero \mathcal{Z} .

- 1: Identificar y trabajar sólo con el subgrafo más pequeño que contiene a \mathcal{X} , \mathcal{Y} , \mathcal{Z} , y sus ancestros.
 - 2: Añadir enlaces no dirigidos entre aquellos nodos que tengan un hijo común.
 - 3: Convertir todos los enlaces dirigidos en enlaces no dirigidos.
 - 4: Si cada camino que conecte una variable de \mathcal{X} con una variable de \mathcal{Y} contiene una variable de \mathcal{Z} , entonces \mathcal{X} es condicionalmente independiente de \mathcal{Y} dado \mathcal{Z} .
-

A veces puede resultar difícil determinar qué es una dependencia causal y qué no lo es. Imagine que tiene los nodos X = “Usar paraguas” e Y = “Está lloviendo”. Se podría decir que cuando uno observa que la gente usa paraguas es bastante probable que esté lloviendo y, en consecuencia, se debería establecer un enlace desde “Usar paraguas” hasta “Está lloviendo”, $X \rightarrow Y$. Esto, sin embargo, desde una perspectiva causal es un tremendo error ya que realmente la relación entre ambas variables es que es el hecho de llover la causa de que la gente use paraguas. El arco causal debería ser desde “Está lloviendo” hasta “Usar paraguas”, $Y \rightarrow X$.

La moraleja del ejemplo es que cuando construya una red causal debe asegurarse de que los enlaces sean realmente causales. Pero se pone de manifiesto algo más. Si en vez de pensar en causa/efecto se piensa en términos de creencia, influencia, dependencia o relevancia tan válido es el enlace $X \rightarrow Y$ como el enlace $Y \rightarrow X$. En efecto, el que la gente use paraguas modifica nuestra credibilidad de que esté lloviendo; pero el que llueva también nos hace pensar que es muy creíble que la gente use paraguas. Entonces, ¿por qué no representar por $X \rightarrow Y$ el hecho de que ambas variables tienen influencia mutua sin importar su causalidad? Cuando ésta es la interpretación de los arcos se habla de **redes bayesianas**.

6.3.2 Redes bayesianas

Las redes bayesianas son estructuras gráficas (grafos dirigidos acíclicos) que permiten modelar el conocimiento incierto sobre un conjunto de variables y razonar con él. Constan de dos componentes:

- Componente cualitativa. Definida por el conjunto de variables (nodos) y el conjunto de arcos (o enlaces) que conectan pares de nodos. La topología del grafo

representa las dependencias entre variables tanto directas, a través de sus arcos, como también mediante los patrones de conexión estudiados para redes causales. Esto significa que el criterio de d-separación de la Definición 6.8 es válido para cualquier red bayesiana.

- Componente cuantitativa. La define un conjunto de valores de probabilidad que indican la fuerza de la conexión entre pares de variables enlazadas. Parece natural que, desde la perspectiva probabilística, se considere que la fuerza de un enlace $X \rightarrow Y$ sea $P(Y|X)$. Claro que si Y tuviera como padres a X y a Z debería considerarse cómo interactúan los dos sobre Y , por lo que se debería especificar $P(Y|X, Z)$.

Formalmente, las redes bayesianas pueden definirse como sigue:

Definición 6.9 (Red Bayesiana). *Una red bayesiana es un grafo dirigido acíclico que representa el conocimiento disponible sobre un conjunto de variables $X_N = \{X_1, X_2, \dots, X_n\}$ que cumple las siguientes condiciones:*

1. *Cada nodo de la red representa una variable, X_i , del conocimiento a modelar.*
2. *La topología de la red representa condiciones de (in)dependencia condicional de las variables del grafo según el criterio de d-separación.*
3. *Cada nodo X con padres $\{U_1, U_2, \dots, U_p\} \subset X_N$ define un conjunto de probabilidades. Más concretamente, para cada configuración (u_1, u_2, \dots, u_p) de los padres, se define una probabilidad condicional con valores $P(x|u_1, u_2, \dots, u_p)$. Es decir, cada nodo X tendrá tantas probabilidades (condicionadas) como configuraciones de valores puedan establecerse con sus padres.*

Analicemos un ejemplo de red bayesiana.

- La parte cualitativa se muestra en la Figura 6.4. Consta de cinco variables booleanas con valores $\{v = \text{Verdad}, f = \text{Falso}\}$ y refleja la siguiente situación. Un estudiante podría obtener su título de carrera (representado por *Carrera*) si aprobara una asignatura (*Aprobar*). Tiene más posibilidades de conseguir un puesto de trabajo (*Trabajo*) si acredita que tiene conocimientos de la asignatura que debe superar. El alumno puede superar la asignatura o estudiando (*Estudia*) o copiando (*Copia*). Además, la topología nos informa de cómo se influyen las variables entre sí: estudiar influye para la consecución de su carrera pero estos dos hechos son independientes si se sabe que aprobó la asignatura (por ser un patrón en serie); conocer que el alumno está trabajando es independiente de que supere su carrera, sabido que suspendió o aprobó la asignatura (al ser un patrón divergente); o que el hecho de no estudiar afecta a nuestra credibilidad de que se haya copiado si aprobó la asignatura (por ser patrón convergente); etc.
- La componente cuantitativa, Figura 6.5, nos muestra, por ejemplo, que se tiene muy poca credibilidad de que el alumno se copie ($P(Co = v) = 0,001$), se esperan buenos resultados de él si estudia ($P(Es = v|Co = f, Es = v) = 0,95$)

y algo mejores si se copia, la probabilidad de que pueda trabajar es diez veces mayor si aprobara la asignatura ($P(Tr = v|Ap = v) = 10P(Tr = v|Ap = f)$), o no conseguirá el título si no aprueba ($P(Ca = v|Ap = f) = 0$). Observar que en este ejemplo sólo es necesario establecer la probabilidad p para la instancia v de cada variable del grafo, dada una configuración de los padres. La probabilidad para la instancia f viene dada por $1 - p$.

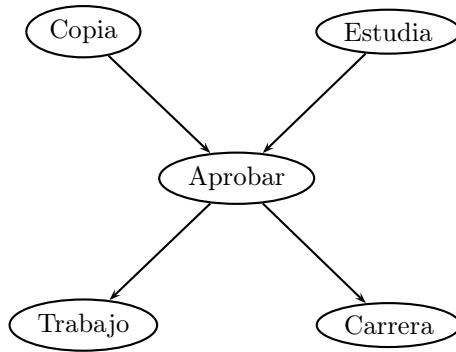


Figura 6.4: Componente cualitativa de una red bayesiana.

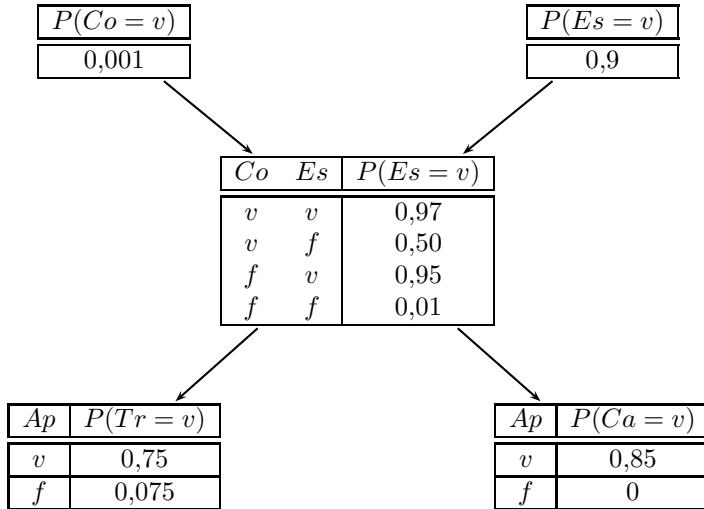


Figura 6.5: Componente cuantitativa de la red bayesiana.

Con este ejemplo, retomemos la Definición 6.9 de red bayesiana. Las condiciones 2 y 3 ponen de relieve la componente cualitativa y cuantitativa de estas representaciones gráficas. La segunda condición establece que el criterio de d-separación que se ha

establecido para redes causales sigue siendo válido en redes bayesianas y por tanto las tres reglas de flujo de información previamente establecidas para los tres patrones de conexión se siguen aplicando. Sin embargo, en este contexto se suele recurrir a la siguiente notación. Si $\langle X, Y | Z \rangle$ denota el hecho de que X es independiente de Y , dado Z , según el criterio de d-separación, entonces:

- Para un patrón en serie y para un patrón divergente se cumple que

$$\langle Y_i, Y_j | X \rangle \quad \text{y} \quad \neg \langle Y_i, Y_j | \emptyset \rangle \quad (6.7)$$

donde X es una variable intermedia entre las variables Y_i e Y_j para el patrón en serie, y es la variable conectora para un patrón divergente. En este caso se habla de **dependencias normales**.

- Para un patrón convergente se cumple que

$$\neg \langle Y_i, Y_j | X \rangle \quad \text{y} \quad \langle Y_i, Y_j | \emptyset \rangle \quad (6.8)$$

donde X es la variable conectora. Estas situaciones reciben el nombre de **dependencias inducidas**.

La tercera condición es una de las grandes ventajas de usar redes bayesianas: representan modelos probabilísticos de una forma mucho más compacta que la distribución conjunta de todas las variables, y esto permite representar probabilidades de un modo computacionalmente tratable. Esta compactación es un ejemplo de una propiedad más general de los sistemas **dispersos**. En un sistema disperso se considera que el sistema está formado por un conjunto de componentes y cada uno interactúa sobre un número (relativamente) pequeño de otras componentes. En estos sistemas suele ocurrir que cada estructura local (o componente) crece en complejidad lineal, y esto es lo que le ocurre a las redes bayesianas. Por ejemplo, suponga que tiene un modelo probabilístico formado por n variables binarias. El número total de valores necesario para establecer la probabilidad conjunta es $2^n - 1$. En otras palabras, los sistemas probabilísticos basados en probabilidades conjuntas crecen en complejidad exponencial. Imagine ahora que el modelo pudiera representarse con una red bayesiana que responda a una conexión en serie. Es muy fácil comprobar que para esta red sería necesario establecer tan sólo $2n - 1$ valores. En general, si cada variable está influida por un número medio de k padres, entonces necesitará especificar como mucho 2^k valores por nodo, por lo que el crecimiento en complejidad es del orden lineal $n2^k$. La reducción en el número de parámetros ha sido realmente drástica y, además, la complejidad pasa de un crecimiento exponencial a uno lineal.

De manera natural surge, por tanto, la siguiente pregunta: partiendo de un modelo de probabilidad ¿cómo se puede obtener una red bayesiana que lo represente? Tal representación es posible si el modelo cumple la condición de Markov, que establece lo siguiente.

Definición 6.10 (Condición de Markov). *Si dispone de una probabilidad conjunta, P , sobre n variables y un grafo cualquiera, G , donde cada nodo representa a cada una de esas variables, entonces P cumple la condición de Markov para el grafo G si para cada variable se verifica que es condicionalmente independiente de sus no-descendientes dados sus padres (según la estructura del grafo).*

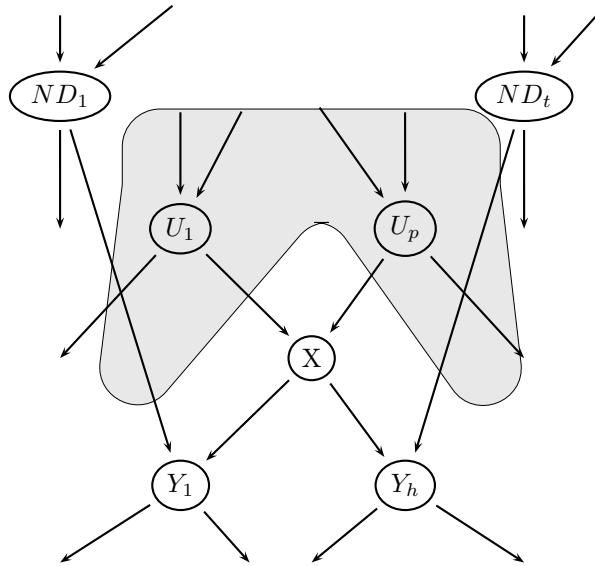


Figura 6.6: *Representación gráfica* de la Condición de Markov: Un nodo es condicionalmente independiente de sus no-descendientes, dados los padres.

En la Figura 6.6 puede ver una representación gráfica de esta condición, donde el nodo X está *aislado* por la zona sombreada (sus padres U s) de sus nodos no-descendientes (los nodos ND s). Una probabilidad P cumpliría la condición de Markov para la Figura 6.6 si se verificara que

$$P(X|ND_1, \dots, ND_t, U_1, \dots, U_p) = P(X|U_1, \dots, U_p)$$

A partir de la Definición 6.10 se puede demostrar el resultado siguiente: si P cumple la condición de Markov según G , entonces P es igual al producto de sus probabilidades condicionales de todos los nodos dados los valores de sus padres. En otras palabras: si dispone de una probabilidad conjunta P y es capaz de construir un grafo para el que se cumpla la condición de Markov, entonces G será una red bayesiana que representará a P . Este resultado tan importante permite establecer el Algoritmo 6.3 de construcción de redes bayesianas que representan a modelos probabilísticos [Pearl, 1988]. Además, el grafo resultante es un I-mapa minimal. Por I-mapa se entiende a todo grafo que represente una probabilidad conjunta verificando la propiedad “ $<X, Y|Z>\Rightarrow I(X, Y|Z)$ ”. Esto es, si dispone de un representación (el grafo) y de un

modelo (la probabilidad conjunta), el grafo es un I-mapa si las independencias entre variables establecidas por el criterio topológico de d-separación corresponden en el modelo probabilístico a independencias condicionadas. Un I-mapa es minimal si al quitar un arco del grafo, el grafo resultante deja de ser un I-mapa.

Algoritmo 6.3 Construcción de una red bayesiana a partir de un modelo de probabilidad.

Entradas: Un conjunto de variables $X_N = \{X_1, X_2, \dots, X_n\}$ y un modelo de probabilístico P sobre X_N .

Resultado: Una red bayesiana sobre X_N .

- 1: Construir un grafo formado sólo por nodos y sin relacionarlos entre sí. Uno por cada variable de entrada.
- 2: Considerar una ordenación arbitraria de las variables. Sin pérdida de generalidad se puede suponer que es X_1, X_2, \dots, X_n .
- 3: **para** $i = 1, 2, \dots, n$ **hacer**
- 4: Definir $X_{T_i} = \{X_1, \dots, X_{i-1}\}$ {Conjunto formado por las variables previas a i .}
- 5: Asignar como padres del nodo X_i al conjunto más pequeño de nodos $X_{F_i} \subseteq X_{T_i}$ que verifiquen la condición:

$$P(X_i|X_{T_i}) = P(X_i|X_{F_i})$$

- 6: Asociar la probabilidad $P(X_i|X_{F_i})$ al nodo X_i .

- 7: **fin para**
-

Es claro que en el Algoritmo 6.3 diferentes órdenes de variables generan estructuras diferentes y que todas ellas representan a la misma probabilidad conjunta. Existen diferentes razones para intentar construir la estructura más compacta posible [Korb y Nicholson, 2004]. Por ejemplo, este factor es decisivo desde el punto de vista de la eficiencia de los procesos de razonamiento pues ésta depende de la factorización de la probabilidad conjunta bajo la hipótesis de independencia condicional [Shachter y otros, 1991]. Así, en la medida de lo posible, intente encontrar siempre la red que tenga un menor número de enlaces. En el caso de redes causales el *orden óptimo* es añadir como nodos raíz a las primeras causas, añadir las variables que están influidas *directamente* por las variables de nivel superior, y continuar hasta llegar a los nodos hoja.

En la práctica, no siempre se parte de una probabilidad conjunta para, posteriormente, obtener una red bayesiana que la represente. En muchas ocasiones se suele trabajar en sentido contrario. Se empieza por una red bayesiana y se debe concluir que el producto de todas las probabilidades condicionadas es una probabilidad conjunta que satisface la condición de Markov para la red. Por fortuna, este resultado también se cumple: Dada una red bayesiana definida por las probabilidades $H = \{P(X_i|X_{F_i})\}_{i=1}^n$, con X_{F_i} el conjunto de nodos padre de la variable X_i , entonces estas probabilidades definen una probabilidad conjunta que viene dada por:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i|x_{F_i}) \tag{6.9}$$

Esta expresión también se conoce como **regla de la cadena para redes bayesianas** y será básica para el cálculo de inferencias como veremos en la sección siguiente.

La justificación del nombre es la siguiente. La **regla de la cadena** del cálculo de probabilidades establece que la probabilidad conjunta sobre n variables puede factorizarse como producto de probabilidades condicionadas según la expresión:

$$P(x_{\sigma(1)}, \dots, x_{\sigma(n)}) = \prod_{i=1}^n P(x_{\sigma(i)} | x_{\sigma(1)}, \dots, x_{\sigma(i-1)}) \quad (6.10)$$

donde σ representa cualquier ordenación sobre los n índices. Observe la semejanza entre (6.9) y (6.10). Hay algo más que semejanza: la Ecuación (6.9) no es más que la aplicación de la regla de la cadena para una ordenación ancestral de los nodos de la red bayesiana para la que se cumple que $X_{F_i} \subseteq \{X_{\sigma(1)}, \dots, X_{\sigma(i-1)}\}$. Es decir, (6.9) es un caso particular de (6.10).

En [Pearl, 1988], [Neapolitan, 1990] o [Castillo y otros, 1996] encontrará muchas más propiedades interesantes que presentan las redes bayesianas ya sea en su versión cualitativa, como modelo que refleja (in)dependencias según ciertos modelos topológicos, o bien en su versión cuantitativa, como modelo probabilístico.

6.4 Razonamiento con redes bayesianas

Uno de los objetivos de los sistemas inteligentes es poder deducir conocimiento nuevo a partir del conocimiento disponible y a la luz de la información que le suministra el usuario. En este apartado se estudia cómo resolver este problema, cuando el conocimiento (incierto) está representado por una red bayesiana sobre un conjunto de variables X_N .

Para abordar el problema, es necesario conocer qué tipo de información puede suministrar el usuario. Es claro que, en este contexto, la información certera hace referencia a los valores de las variables. De hecho, ya se ha hecho referencia a que el valor conocido de una variable se denomina **evidencia** y se ha comentado su repercusión en los tres patrones topológicos básicos. Realmente, en ese estudio, se estaba considerando la evidencia como un hallazgo exacto en el que se descubre que cierta variable X toma un valor concreto x , y notado como $X = x$. A este tipo de descubrimiento también se le denomina **evidencia específica**. Por ejemplo, si descubre que un alumno se ha copiado entonces $Copia = v$ es una evidencia específica.

Sin embargo, en otras ocasiones, la evidencia se manifiesta en términos de qué valores no son posibles en el estado actual de la variable. En este caso se habla de **evidencia negativa**. Una situación se produce cuando se dice que una variable X no se encuentra en una caso concreto x ; es decir, $X \neq x$ e implica que la variable puede tomar cualquiera de sus otros valores. Otra situación diferente responde a afirmar que una variable X se puede encontrar o en el caso x_A o en el caso x_B ; esto es, $(X = x_A \vee X = x_B)$ y hace que el resto de los valores no sean posibles.

Pero también hay otro tipo de evidencia, la **evidencia virtual**. Aparece cuando el valor de una variable no puede establecerse de forma categórica pero, en su lugar, se establece un grado de creencia en el valor que se considera observado. Por ejemplo, cuando un colega del trabajo le dice que *cree* haber visto que le estaban robando el

coche, no está estableciendo un valor bien definido. Si le dice que sólo está seguro en un 80 % nos manifiesta una evidencia virtual. En estos casos la evidencia se suministra vía funciones de **verosimilitud**. Cuando la verosimilitud es 1, entonces dispone de evidencia específica. En [Korb y Nicholson, 2004] tiene una buena referencia sobre el tratamiento de la evidencia virtual en redes bayesianas.

Una vez conocida la evidencia, se debe plantear qué tipo de razonamiento hay que aplicar. En redes bayesianas son dos los tipos de razonamientos más ampliamente utilizados⁴: la inferencia y la abducción.

Inferencia probabilística. La inferencia probabilística, también conocida como inferencia de la evidencia o actualización de la creencia, se plantea determinar el conocimiento referente a un conjunto de variables de interés, $X_I = \{x_i | i \in I\}$, teniendo en cuenta que se conoce el estado de algunas de ellas, $(X_E = e) = \{x_j = e_j | j \in E\}$. Esto es, la inferencia probabilística se plantea la obtención de los valores

$$Bel(x_I) = P(x_I | x_E = e).$$

Mención especial tienen los siguientes casos en el contexto de *redes causales*. Si la evidencia está disponible en las causas y se está interesado en la modificación de creencias en sus efectos (siguiendo el sentido de los arcos) se habla de **razonamiento predictivo**. Cuando la evidencia está disponible sólo en los síntomas y se está interesado en la repercusión que produce en sus causas (siguiendo el sentido opuesto de los arcos) se habla de **razonamiento diagnóstico**.

Interpretación de evidencias. Dada una probabilidad P sobre las variables X_N y con valores de evidencia e , cualquier asignación de valores de la variable X_N que sea consistente con e recibe el nombre de **explicación**, extensión o interpretación de e . Obviamente, para que $X_N = \{X_{N-E}, X_E\}$ sea una explicación de e , el valor de X_E deben de ser igual a e .

Es claro que dada una evidencia, e , se puede encontrar una multitud de explicaciones para que se cumpla e , pero ¿de todas ellas cuál es más creíble? El problema que ahora se plantea es encontrar la explicación x_N^* que maximiza la probabilidad condicionada $P(x_N | e)$. El valor $X_N = x_N^*$ recibe el nombre de **explicación más probable** (o MPE⁵) de la evidencia e y verifica:

$$P(x_N^* | e) = \max_{x_N} P(x_N | e).$$

En este tipo de razonamiento, otra tarea es la determinación de la **máxima hipótesis a posteriori** (o MAP⁶) que trata de encontrar la MPE para sólo algunas variables de la red. Es decir, dado un conjunto de variables $X_I \subset X_N$, consideradas como hipótesis, el objetivo es encontrar una instancia x_I^* de X_I tal que

$$P(x_I^* | e) = \max_{x_I} P(x_I | e).$$

⁴En lo que sigue se supondrá que sólo se dispone de evidencia específica.

⁵MPE, acrónimo de Most Probable Explanation.

⁶MAP, acrónimo de Maximum Aposteriori Hypothesis.

La resolución de estos problemas de deducción no son difíciles desde un punto de vista teórico pues bastará calcular la marginal de las variables de interés (para la inferencia) o calcular un máximo (para la abducción) a partir de una tabla de valores de probabilidad. Sin embargo, desde el punto de vista computacional, todos estos problemas son NP-duros.

Existen distintas técnicas para abordar estas tareas. De las más estudiadas y pioneras en este tema son las basadas en el **envío de mensajes**. Estas técnicas se basan en aprovechar las condiciones de dependencias e independencias que existen entre las variables. Intuitivamente puede explicarse de la manera siguiente. Teniendo en cuenta que una probabilidad no es más que una tabla informativa sobre las variables, entonces se pueden utilizar las condiciones de independencias para formar pequeños *trozos* de información obtenidos a partir de la tabla informativa y que involucren a unas pocas variables para, posteriormente, utilizar las dependencias para transmitir la información de los distintos grupos entre sí pasando mensajes de unos a otros. Al final del proceso, cada grupo tendrá su propia información junto con todos los mensajes recibidos por los demás grupos. Bastará entonces recurrir al grupo más adecuado para obtener la información requerida y de interés. Cuando el grafo no presenta ningún tipo de ciclo (no dirigido) cada grupo puede estar formado por sólo una variable. Cuando la red bayesiana presenta ciclos (no dirigidos) entonces se construyen grupos especiales [Huang y Darwiche, 1996].

En este capítulo se mostrarán los fundamentos de los algoritmos basados en **eliminación de variables** ya que desde un punto de vista docente proporciona un marco más unificado y sencillo, aunque no siempre sea el más eficiente. Puede consultar [Li y D'Ambrosio, 1994], [Zhang y Poole, 1996] y [Dechter, 1999] para un estudio más detallado de la técnica que aquí se muestra.

6.4.1 Inferencia probabilística

El cálculo de $P(X_I|X_E = e)$ puede llevarse a cabo aplicando simplemente la regla de Bayes, Ecuación (6.4), y que puede reescribirse como sigue aplicando la regla de la cadena para redes bayesianas:

$$P(x_I|x_E = e) = \frac{P(x_I, x_E = e)}{P(x_E = e)} \quad (6.11)$$

$$= \frac{\sum_{x_{N \setminus I \cup E}} \left[\prod_{i=1}^n P(x_i|x_{F_i}) \right]_{|x_E=e}}{P(x_E)_{|x_E=e}} \quad (6.12)$$

donde $f(Z)_{|Z=z}$ indica que primero se deben realizar los cálculos que indique f y posteriormente particularizar al caso $Z = z$.

Por **métodos exactos** se conocen al conjunto de técnicas que aplican directamente la Ecuación (6.11) o cualquier otra expresión derivada como la Ecuación (6.12). Fueron los primeros en desarrollarse. El problema de la inferencia mediante técnicas exactas en redes bayesianas entra en la categoría de problemas NP-duros [Cooper, 1990]. Esto quiere decir que la complejidad de la resolución del problema crece de

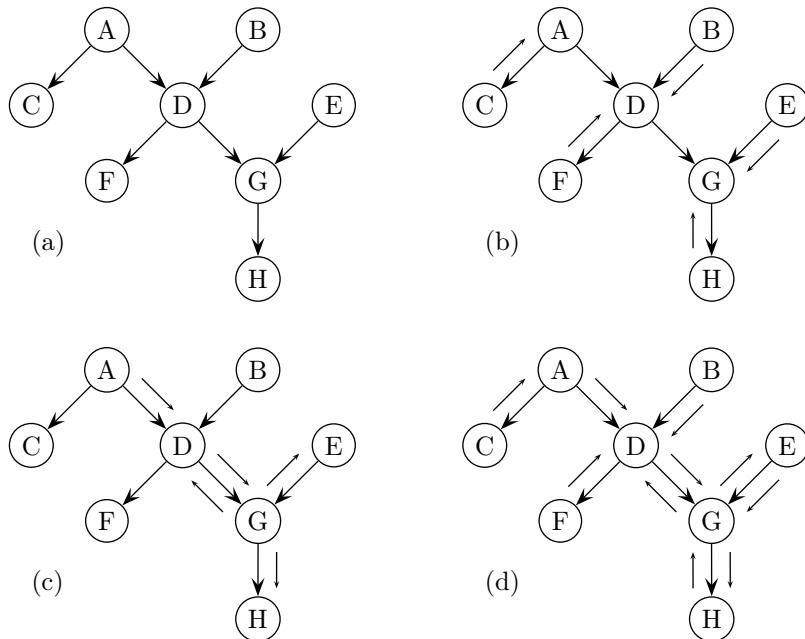
manera exponencial conforme se aumenta el número de casos de la red. Es decir, sea cual sea el algoritmo exacto que diseñe puede encontrar situaciones en las que el algoritmo resulta ineficiente. Como alternativa a los métodos exactos surgieron los **métodos aproximados**. Estos métodos intentan resolver aquellos casos en los que los métodos exactos resultaban claramente inefficientes pretendiendo resolver esas situaciones en un tiempo más razonable, aunque conlleve perder cierta precisión en los resultados. Desgraciadamente, la inferencia aproximada también es un problema NP-duro cuando se requiere de una precisión determinada [Dagum y Luby, 1993].

6.4.1.1 Inferencia exacta

En 1986 J. Pearl desarrolló un método de modificación de las distribuciones de los nodos de un árbol, generalizando su desarrollo a poliárboles [Pearl, 1988]. Fue el primer algoritmo que se construyó para abordar el problema de la inferencia. Se basa en el envío de mensajes respetando la estructura original topológica del grafo. Es decir, los grupos que reciben o envían mensajes están formados por sólo una variable y la conexión entre los grupos (variables) son los propios arcos del grafo original. Está demostrado que el algoritmo de J. Pearl es el más eficiente que puede diseñarse para la inferencia en poliárboles.

La Figura 6.7 muestra un ejemplo de cómo funciona y está inspirado en el presentado en [Castillo y otros, 1996]. Básicamente, cuando en un nodo se modifica la información asociada, ésta se traspasa a los nodos vecinos a través de los arcos que los unen; éstos a su vez pasan la nueva información junto con la que ya tenían a aquellos nodos vecinos aún no modificados y así sucesivamente. Conviene observar que un nodo no envía un mensaje de información a un vecino suyo hasta que no ha recibido toda la información referente a todo el grafo, salvo la del subgrafo que determina el vecino al que se le quiere enviar la información. Por ejemplo, en la Figura 6.7.(c), el nodo D recibe toda la información del subgrafo definido por $\{A, C\}$ a través del arco $A \rightarrow D$ y en la Figura 6.7.(e), el nodo A recibe toda la información del subgrafo definido por todos los nodos, salvo $\{A, C\}$ a través del mismo arco. Note que D no envía un mensaje a A hasta que D no ha recibido la información de sus otros vecinos B , F y G .

Por desgracia, el algoritmo no se puede generalizar a estructuras que presenten ciclos (no dirigidos), por lo que se ha dedicado mucho esfuerzo en estudiar generalizaciones del algoritmo para poliárboles o en crear métodos alternativos. Lo curioso es que, gráficamente, en la mayoría de estos algoritmos se producen modificaciones de la estructura original del grafo para generar una nueva *estructura arbórea* donde cada nodo está formado ahora por varios nodos del grafo original y, computacionalmente, los cálculos que se realizan tienen una *interpretación de envío de mensajes*. Algunas de tales técnicas son las basadas en árboles de grupos o *clusters*, siendo los más utilizados los basados en *cliques*. En la Figura 6.9 puede ver un ejemplo de una de esas posibles estructuras. Se trata de un árbol de cliques que puede construirse a partir del grafo de la Figura 6.7.(a). A partir de este árbol de grupos se pueden aplicar algoritmos de envío de mensajes adaptados a este tipo de grafos. Para más detalles sobre cómo construirlos y cómo propagar los mensajes puede consultar [Lauritzen y Spiegelhalter, 1988], [Shenoy, 1989] y [Jensen y otros, 1990].



- (a) Poliárbol de partida.
 (b) En la primera iteración algunos nodos padres y todos los nodos hojas mandan información a los demás nodos. Los nodos que mandan información son B , C , E , F y H . Los nodos A , D y G reciben información.
 (c) Los nodos A , D y G detectan que han recibido información. Intentan transmitir nuevos mensajes a sus vecinos, si pueden. D manda un mensaje a G , pero no puede mandar un mensaje a A .
 (d) Muestra todos los mensajes que se han mandado después de dos iteraciones del algoritmo. Todos los nodos salvo D y A ya han mandado todos lo mensajes que pueden enviar.

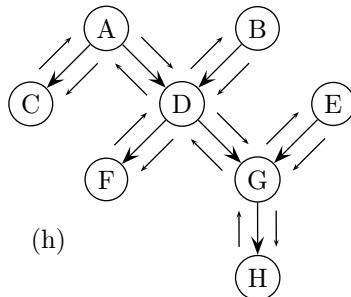
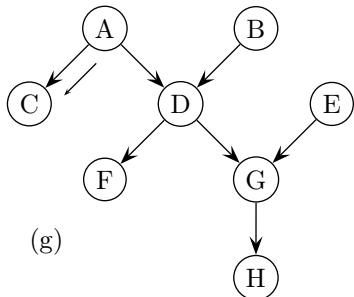
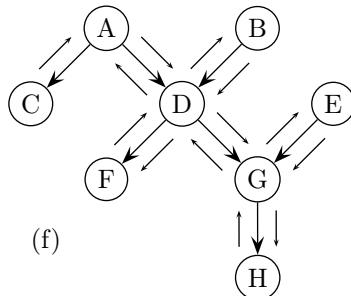
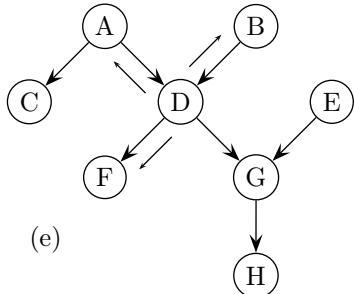
(continúa en la Figura 6.8)

Figura 6.7: Envío de mensajes en un poliárbol (I).

Aquí nos centraremos en los fundamentos del grupo de técnicas basadas en eliminación de variables que permiten hacer inferencia sobre cualquier tipo de red, aunque no se tienen garantías de que lo hagan siempre de la forma más eficiente posible.

Entrando en materia, observe que calcular la distribución $P(X_I|e)$ es básicamente el mismo problema que calcular la distribución $P(X_I)$. La diferencia es que para el primero se conoce el estado concreto de algunas variables (el valor e es conocido). Así, nuestro objetivo es desarrollar un algoritmo que permita calcular los valores $P(x_I)$ para, posteriormente, hacer las modificaciones oportunas para el cálculo de $P(x_I|e)$.

(continuación de la Figura 6.7)



- (e) D envía los mensajes que le faltan.
 - (f) Mensajes enviados después de tres iteraciones.
 - (g) A ya recibió en la iteración anterior toda la información disponible en el subgrafo formado por todos los nodos menos A y C . A ya está en condiciones de mandar información a C .
 - (h) Ya se han enviado todos los mensajes.
- En este punto cada nodo contiene su distribución marginal: $P(A)$, $P(B)$, ...

Figura 6.8: Envío de mensajes en un poliárbol (y II).

Para ver cómo se puede calcular la marginal de un conjunto de variables se considera el siguiente ejemplo. Dada una distribución sobre tres variables, por la regla de la cadena se sabe que:

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

Si se desea calcular la distribución marginal de x_1 entonces se debe calcular:

$$f(x_1) = \sum_{x_2, x_3} P(x_1, x_2, x_3)$$

O equivalentemente:

$$f(x_1) = \sum_{x_2} \sum_{x_3} P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

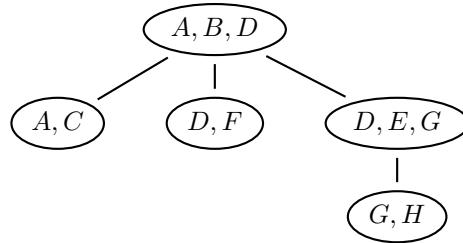


Figura 6.9: Árbol de cliques

Aplicando la propiedad distributiva:

$$f(x_1) = P(x_1) \left\{ \sum_{x_2} P(x_2|x_1) \left[\sum_{x_3} P(x_3|x_1, x_2) \right] \right\} \quad (6.13)$$

Recuerde que el cálculo de marginales puede leerse como *eliminar por sumas*, de la probabilidad conjunta, todas las variables que no son de nuestro interés (véase la Ecuación (6.2)). En nuestro caso, el cálculo de la expresión (6.13) supone eliminar las variables X_2 y X_3 ; pero el orden de eliminación no es cualquiera, sino el que viene determinado por el orden de preferencia de corchetes y llaves. En primer lugar se comienza eliminando por sumas la variable X_3 y, después, la variable X_2 . Más concretamente, el cálculo de (6.13) puede establecerse en los siguientes tres pasos:

- El primer proceso de eliminación, el de X_3 , consiste en seleccionar, de entre las tres probabilidades condicionadas, todas las funciones que contengan X_3 , combinarlas (en este caso sólo hay una) y el resultado marginalizarlo en todas las variables de la función resultante de la combinación menos en la variable X_3 . En fórmulas, la eliminación es calcular

$$f_1(x_1, x_2) = \sum_{x_3} P(x_3|x_1, x_2)$$

y sustituir todas las funciones que contengan X_3 por f_1 . Esto es, (6.13) es equivalente a:

$$f(x_1) = P(x_1) \left\{ \sum_{x_2} P(x_2|x_1) f_1(x_1, x_2) \right\} \quad (6.14)$$

- Posteriormente, en (6.14) se elimina por sumas la variable X_2 . Ahora, el proceso de eliminación consiste en seleccionar de entre las funciones $P(x_1)$, $P(x_2|x_1)$ y $f_1(x_1, x_2)$ aquellas que contengan X_2 en su dominio. En este caso, se seleccionan a $P(x_2|x_1)$ y $f_1(x_1, x_2)$ para combinarlas y marginalizar el resultado en todas las variables de la función resultante de la combinación, menos en la variable X_2 . Es decir, la eliminación de X_2 es calcular:

$$f_2(x_1) = \sum_{x_2} P(x_2|x_1) \cdot f_1(x_1, x_2)$$

y sustituir todas las funciones que contenían a X_2 por f_2 . Así, la Ecuación (6.14) se transforma en

$$f(x_1) = P(x_1) \cdot f_2(x_1) \quad (6.15)$$

- La expresión (6.15) nos dice que, una vez eliminadas las variables X_2 y X_3 (que no son de nuestro interés), se obtienen sólo funciones definidas en las variables en las se está interesado, en nuestro caso la variable X_1 . Es más, la marginal buscada se obtiene como combinación de todas esas funciones.

Algoritmo 6.4 Eliminación por sumatoria de la variable X_i de un conjunto H de funciones.

Entradas: X_i , una variable. H , un conjunto de funciones donde alguna se define, al menos, sobre X_i

Resultado: Un nuevo conjunto de funciones que *condensa* la misma información que H , pero sin la variable X_i .

1: Seleccionar aquellas funciones h de H que contienen la variable X_i de interés. Llamar H_i al conjunto de dichas funciones.

2: Calcular f_i como el producto de todas las funciones de H_i : $f_i := \prod_{h_i \in H_i} h_i$.

3: Redefinir f_i como el resultado de sumar en todos los valores de la variable X_i . Es decir,

$$f_i := \sum_{x_i} f_i = \sum_{x_i} \prod_{h_i \in H_i} h_i.$$

4: devolver $H := [H - H_i] \cup \{f_i\}$.

Con esta idea se construye el Algoritmo 6.5. Se basa en la regla de la cadena para redes bayesianas y permite calcular distribuciones marginales basadas en la eliminación de variables de la red. El algoritmo se fundamenta a su vez en el Algoritmo 6.4, que calcula la función resultante del proceso de eliminación de una variable de un conjunto de funciones.

La eficiencia del Algoritmo 6.5 se encuentra en el paso 3:, la selección del índice i . Un caso algo extremo le permitirá visualizar la situación. Si en la conexión divergente de la Figura 6.1 estuviese interesado en la marginal de la variable Y_n , ésta puede calcularse mediante $(n - 1)!$ ordenaciones posibles de eliminación de variables. En particular, la secuencia de eliminación $\sigma = (1, 2, \dots, n - 1, 0)$, donde el índice 0 corresponde al nodo conexión X , conduce al cálculo:

$$f(y_n) = \sum_x P(x) P(y_n|x) \left\{ \sum_{y_{n-1}} P(y_{n-1}|x) \cdots \left[\sum_{y_2} P(y_2|x) \left(\sum_{y_1} P(y_1|x) \right) \right] \cdots \right\}$$

Pero también es válida la secuencia $\sigma' = (0, 1, 2, \dots, n - 1)$.

$$f(y_n) = \sum_{y_{n-1}} \cdots \sum_{y_1} \sum_x P(x) P(y_1|x) P(y_2|x) \cdots P(y_n|x)$$

Si las variables son binarias, el orden σ necesita en cada paso calcular cuatro valores en la multiplicación de las funciones, salvo para la eliminación de Y_1 y de X . En total requiere de $4n - 2$ multiplicaciones. Sin embargo, para el orden σ' es necesario calcular

2^n valores en el primer paso y conlleva un total de $2^{n+2} - 2^2$ multiplicaciones. Es claro que tanto por el número de operaciones a desarrollar (tiempo) como por el espacio de memoria requerido, el orden σ' es claramente inefficiente y σ el más adecuado. El ejemplo pone de manifiesto que es necesario establecer *órdenes buenos* para intentar ser algo más eficientes.

Algoritmo 6.5 Cálculo de distribuciones marginales en una red bayesiana.

Entradas: X_I , conjunto de variables de interés. H , el conjunto unión de todas las probabilidades condicionadas de la red.

Resultado: La probabilidad marginal $P(X_I)$.

- 1: Si N es el conjunto de índices de todas las variables de la red e I es el conjunto de índices de las variables de interés, definir $J = N - I$.
 - 2: **mientras** $J <> \emptyset$ **hacer**
 - 3: *i* = Seleccionar(J, H) {Seleccionar un *i* de J . Llamar, por ejemplo, al Algoritmo 6.6}
 - 4: $H = Eliminar(X_i, H)$. {Eliminar X_i de H . Llamar al Algoritmo 6.4}
 - 5: $J = J - \{i\}$. {Quitar *i* de J }
 - 6: **fin mientras**
 - 7: **devolver** la combinación de todas las funciones de H .
-

Caben varias posibilidades para establecer el orden de eliminación. El problema no es fácil y de hecho coincide con el problema de la tringulación de grafos, que es un problema NP-duro [Wen, W.X., 1990]. Las mismas técnicas estudiadas en el problema de la triangulación de grafos son aplicables en este contexto, si bien se han diseñado algunas técnicas centradas en el problema de las redes bayesianas (tome [Cano y Moral, 1995] como referencia). Un grupo de éstas son las *heurísticas determinísticas* que se basan en el principio de “*el siguiente nodo a eliminar es el que minimiza f()*”. Por cada función $f()$ que defina, dispone de un nuevo criterio de ordenación. El problema, claro está, es conseguir que $f()$ produzca una secuencia eficiente de eliminación por sumas. Un criterio sencillo que, en general, produce buenos resultados es seleccionar como siguiente variable a eliminar aquella tal que la función resultante de la combinación de todas las funciones que la contienen tenga el menor número de valores. La idea que subyace detrás de este criterio es que si en etapas iniciales no es deseable la eliminación de una variable porque produce un alto consumo de memoria, es, posiblemente, por compartir dominio en muchas funciones; es decir, el alto consumo parece deberse a la participación de bastantes variables. Cabe esperar que si se deja para el final de la secuencia de la eliminación, para entonces es posible que se hayan eliminado algunas de esas otras variables poco deseables y que la función resultante de la combinación sea entonces mucho más pequeña. Este proceso de selección es el que se muestra en el Algoritmo 6.6.

En este punto ya dispone de un sistema completo de inferencia probabilística en redes bayesianas (Algoritmo 6.5); pero sólo es capaz de calcular probabilidades marginales. Queda, por tanto, readaptar el proceso de inferencia para contemplar la existencia de variables observadas con el fin de calcular probabilidades a posteriori según la regla de Bayes. Basta observar la Ecuación (6.11) y analizar sus componentes.

Algoritmo 6.6 Seleccionar la variable a eliminar.

Entradas: H , un conjunto de funciones. X_J , conjunto de variables sobre las que se definen las funciones de H .

Resultado: Una variable de X_J

- 1: Definir H' como el conjunto de todas las funciones y X_J el conjunto de variables sobre las que se definen las funciones de H .
- 2: Establecer \min al mayor valor posible y $variable = desconocida$.
- 3: **para cada** $i \in J$ **hacer**
- 4: Seleccionar aquellas funciones h de H que contienen a la variable X_i de interés. Llamar H' al conjunto de dichas funciones.
- 5: Sea X_T el conjunto de variables sobre las que se definen las funciones de H' .
- 6: Calcular

$$peso = \prod_{X \in X_T} c(X)$$

donde $c(X)$ indica el número de casos de la variable X .

- 7: **si** $peso < \min$ **entonces**
 - 8: $min = peso$
 - 9: $variable = i$
 - 10: **fin si**
 - 11: **fin para**
 - 12: **devolver** $variable$ {Devuelve el índice de la variable que tiene menor coste (peso).}
-

En el denominador, la expresión $P(X_E = e)$ no es más que un número: la constante de normalización, que se puede obtener a partir del numerador $P(X_I, X_E = e)$ como (véase la sección 6.2.5):

$$P(X_E = e) = \sum_{x_I \in X_I} P(X_I, X_E = e)$$

El numerador, $P(X_I, X_E = e)$, es el término realmente importante. En principio, no hay problema en aplicar el Algoritmo 6.5 para el cálculo $P(X_I, X_E)|_{X_E=e}$; es decir, calcular la distribución marginal $P(X_I, X_E)$ y posteriormente restringir la tabla de valores a aquellos en los que $X_E = e$. Sin embargo, de calcularse así, se estarían *arrastrando* algunos valores de X_E que no son de interés. A saber, los valores de X_E que son distintos de e . Un modo de ser más eficiente es añadir una distribución degenerada de probabilidad por cada variable observada:

$$\delta_{e_j}(x_j) = \begin{cases} 1 & \text{si } x_j = e_j \\ 0 & \text{en otro caso} \end{cases} \quad \forall j \in E \quad (6.16)$$

y procesarlas antes de comenzar el algoritmo de eliminación. Es decir, redefinir para cada variable observada, $j \in E$, las distribuciones de probabilidad de la red en las que X_j aparece según la expresión:

$$P(X_i|X_{F_i}) := P(X_i|X_{F_i}) \prod_{j \in E} \delta_{e_j}(x_j)$$

Por ejemplo, si su red bayesiana tuviese la distribución $P(B|A, C)$ y dispusiera de las evidencias $\{B = b_0, C = c_0\}$, su función de distribución entraría en el algoritmo

de inferencia cuando elimine la variable A y la función resultante estará definida al menos en B y en C . Lo que se propone es que realmente no necesita todos los valores informativos de $P(B|A, C)$ sino sólo los valores $P(B = b_0|A, C = c_0)$ para la inferencia o, lo que es lo mismo, puede redefinir su distribución como:

$$P(b|a, c) := P(b|a, c)\delta_{b_0}(b)\delta_{c_0}(c) = \begin{cases} P(b_0|a, c_0) & \forall a \text{ si } b = b_0, c = c_0 \\ 0 & \text{si } b \neq b_0, c \neq c_0 \end{cases}$$

Notar que la función así redefinida ya no es una distribución de probabilidad y que, realmente, es una función unidimensional dependiente sólo de A . Esta función deberá incluirse cuando A vaya a ser eliminada pero, ahora, la función resultante no dependerá ni de B ni de C : acaba de reducir la dimensionalidad del problema.

Con la consideración de todas estas sutilidades se obtiene el Algoritmo 6.7 que permite obtener distribuciones a posteriori sobre cualquier conjunto de variables no observadas de la red basado en el criterio de eliminación de variables.

Algoritmo 6.7 Cálculo de distribuciones a posteriori en una red bayesiana.

Entradas: X_I , conjunto de variables de interés. Evidencias específicas e del conjunto de variables X_E . H , el conjunto unión de todas las probabilidades condicionadas de una red bayesiana.

Resultado: La probabilidad condicionada $P(X_I|X_E = e)$.

1: Instanciar el conjunto de variables observadas X_E al valor e .

2: **para cada** $l \in E$ **hacer**

3: Definir

$$P'(X_i|X_{F_i}) := P(X_i|X_{F_i}) \prod_{j \in E} \delta_{e_j}(x_j)$$

 para cada función $P(X_i|X_{F_i})$ tal que $l \in \{i\} \cup F_i$.

4: Redefinir $H := H \cup \{P'(X_i|X_{F_i})\} \setminus \{P(X_i|X_{F_i})\}$.

5: **fin para**

6: Asignar a $P(X_I, X_E = e)$ los valores resultantes de aplicar Algoritmo-6.5(X_I, H).

7: **devolver** $P(X_I|X_E = e) = \frac{P(X_I, X_E = e)}{\sum_{x_I} P(X_I = x_I, X_E = e)}$.

6.4.1.2 Inferencia aproximada

Los métodos exactos se vuelven inefficientes cuando las redes están formadas por miles de nodos y estos se encuentran muy relacionados entre sí. Una alternativa son los métodos aproximados. La idea de estos algoritmos es resolver esos casos peores empleando menor tiempo computacional y recurriendo a técnicas de Monte Carlo, aunque se pierda exactitud en la precisión. Aunque la resolución del problema de la inferencia por técnicas de simulación también es NP-duro, su uso nos permite aumentar el conjunto de problemas resolubles.

En muchas situaciones complicadas se pueden calcular valores aproximados a las auténticas probabilidades de algunos sucesos mediante técnicas de **simulación**. Una de ellas consiste en desarrollar un experimento y observar los resultados que produce. El conjunto de resultados generados por el experimento después de M realizaciones recibe el nombre de *muestra de tamaño M* y cada resultado, x , recibe el nombre

de *individuo*, realización o dato, y el número M el de *tamaño de la muestra*. Si en la muestra el individuo x se presenta $N(x)$ -veces, la auténtica probabilidad de x se aproxima por su frecuencia relativa $\frac{N(x)}{M}$. Cuanto mayor sea el tamaño de la muestra, mejor será la aproximación.

Una **técnica de Monte Carlo** es una técnica de simulación que permiten obtener muestras, supuesto que sus individuos están modelados de acuerdo a ciertas distribuciones de probabilidad, mediante la generación de números aleatorios. El método de inversión es una de tales técnicas y se muestra en el Algoritmo 6.8. Cada vez que aplique el metodo de inversión obtendrá un individuo de la muestra.

Algoritmo 6.8 Método de inversión.

Entradas: Una variable X con m -valores. Una probabilidad P sobre X con valores $P(X = x_i) = p_i$, $i = 1, 2, \dots, m$

Resultado: Un individuo x según la probabilidad P .

- 1: Generar un número aleatorio rnd entre 0 y 1.
- 2: Definir $F = p_1$ y $i = 1$
- 3: **mientras** ($i \leq m$) y ($F < rnd$) **hacer**
- 4: $i = i + 1$
- 5: $F = F + p_i$
- 6: **fin mientras**
- 7: $x = x_i$
- 8: **devolver** x

El problema en redes bayesianas es que aplicar el método de inversión requiere calcular la distribución conjunta y eso es precisamente lo que no se quiere hacer porque se supone que es de difícil manejo. En la práctica, cuando las probabilidades originales son de difícil tratamiento, lo que se hace es utilizar una distribución modificada que resulte más sencilla de manejar para asignar, según esa nueva distribución, un peso o importancia del dato muestreado. El promedio de todos los pesos asignados a un individuo de la muestra permite obtener una aproximación de la auténtica probabilidad del individuo. Formalmente, el proceso en redes bayesianas es el siguiente.

Considere un conjunto de variables $X = \{X_1, X_2, \dots, X_n\}$. Recuerde que una instancia de sus valores se indica por $x = (x_1, \dots, x_n)$ para representar que la variable X_i toma el valor x_i . Si I es un subconjunto de índices en $\{1, 2, \dots, n\}$, se indica por $x^{\downarrow I}$ a la acción de *considerar sólo las componentes* de la instancia x con índices en I . Por ejemplo, sobre cuatro variables, $x^{\downarrow \{1,3\}} = (x_1, x_2, x_3, x_4)^{\downarrow \{1,3\}}$ representa a los valores (x_1, x_3) . Con esta notación y usando la función definida en (6.16), puede expresarse el teorema de Bayes para redes bayesianas, Ecuación (6.12), como:

$$P(x_I | x_E = e) = \frac{P(x_I, e)}{P(e)} = \frac{\sum_{x: x^{\downarrow I} = x_I} P(x, e)}{P(e)} \quad (6.17)$$

donde

$$P(x, e) = \left[\prod_{i=1}^n P(x^{\downarrow i} | x^{\downarrow F_i}) \right] \left[\prod_{j \in E} \delta_{e_j}(x^{\downarrow j}) \right] \quad (6.18)$$

Si P^* es una distribución más fácil de manejar que P , y teniendo en cuenta que para la inferencia sólo el numerador de la Ecuación (6.17) es relevante, puede escribirse:

$$P(x_I, e) = \sum_{x: x^{\downarrow I} = x_I} P(x, e) = \sum_{x: x^{\downarrow I} = x_I} \frac{P(x, e)}{P^*(x)} P^*(x) = E_{P^*} \left[\frac{P(x, e)}{P^*(x)} \right]$$

Esta expresión dice que el valor esperado de la variable $\frac{P(x, e)}{P^*(x)}$ es la probabilidad buscada si X se rige por P^* . La función P^* recibe el nombre de **distribución de muestreo**. En la práctica, si se generan M -datos, $x^{(t)}$, según la distribución P^* , entonces se puede obtener la siguiente estimación de la marginal:

$$P(x_I, e) \approx \hat{P}(x_I, e) = \frac{1}{M} \sum_{t \in T} \frac{P(x^{(t)}, e)}{P^*(x^{(t)})} = \frac{1}{M} \sum_{t \in T} w_t \quad (6.19)$$

donde

$$w_t = \frac{P(x^{(t)}, e)}{P^*(x^{(t)})} \quad (6.20)$$

El valor w_t recibe el nombre de **peso**, importancia o *score* del individuo t , y T es el subconjunto de superíndices de aquellos individuos que son coherentes con las restricciones establecidas: si $t \in T$, entonces se cumple que $x^{(j)^{\downarrow I}} = x_I$ y $x^{(j)^{\downarrow E}} = e$.

Algoritmo 6.9 Muestreo por importancia.

Entradas: X_I , conjunto de variables de interés. Evidencias específicas e del conjunto de variables X_E . Las probabilidades condicionadas de una red bayesiana. Una distribución de muestreo P^* . El tamaño de la muestra M .

Resultado: Una estimación de la probabilidad condicionada $P(X_I|X_E = e)$.

- 1: **para** $t = 1, 2, \dots, M$ **hacer**
 - 2: Generar un individuo $x^{(t)}$ a partir de P^* . {P.e. con el Algoritmo 6.8}
 - 3: Calcular su peso según la Ecuación (6.20).
 - 4: **fin para**
 - 5: **para** cada instancia x_I **hacer**
 - 6: Calcular $\hat{P}(x_I, e)$, según la Ecuación (6.19).
 - 7: **fin para**
 - 8: Normalizar los valores $\hat{P}(x_I, e)$ para obtener $\hat{P}(x_I|e)$.
 - 9: **devolver** $\hat{P}(X_I|e)$ {Como el auténtico valor de $P(X_I|e)$ }.
-

Cuando la elección de un individuo no depende de los elegidos previamente (cuando son independientes), el esquema de simulación que se acaba de presentar recibe el nombre de **muestreo por importancia** (véase el Algoritmo 6.9). La construcción de la distribución de muestreo puede construirse de muchos modos, pero tiene una restricción básica: P^* debe de ser capaz de muestrear todas las posibles configuraciones; es decir, $P^*(x) = 0$ solo si $P(x) = 0$ para evitar indefinición. Se demuestra que el estimador $\hat{P}(x_I, e)$ es un estimador insesgado de $P(x_I, e)$ y que permite reducir la varianza de la estimación cuanto más parecida sea P^* a P y tal que los individuos generados produzcan que los pesos w sean lo más constantes posibles. Puede consultar [Rubinstein, 1981] o [Law, 1991] para más detalles teóricos sobre esta técnica de muestreo.

Se le ha mostrado este esquema general porque agrupa técnicas importantes de inferencia aproximada en el contexto de las redes bayesianas y es un referente en el diseño de nuevos algoritmos. Las distintas técnicas que usan este esquema se diferencian entre sí en la elección de cada una de las tres componentes que constituyen un método de simulación:

- La distribución de muestreo, utilizada para generar la muestra.
- Un método de generación de configuraciones según la distribución de muestreo.
- La fórmula utilizada para calcular los pesos.

Sepa también que no todo se basa en el muestreo por importancia y existen muchas otras técnicas (véase [Martin, 1996] para un marco general).

Aunque son muchos los métodos aproximadas de inferencia que se han diseñado en los últimos años, las técnicas pioneras que destacamos pueden incluirse en alguno de los grupos siguientes:

Muestreo hacia adelante: Estas técnicas consideran que cada individuo es independiente de los generados y las componentes que configuran cada individuo se obtienen según un cierto orden de los nodos del grafo. De este grupo se comentarán las técnicas de **muestreo lógico y ponderación de la versosimilitud**.

Muestreo basado en cadenas de Markov: Estas técnicas parten de una instancia de todas las variables de la red. En cada iteración se modifica la instancia de cada variable según cierta distribución de muestreo. Aquí, en principio, no importa el orden considerado en las variables y claramente la configuración de cada individuo depende del previamente generado. De este grupo se comentará la técnica de **simulación estocástica**.

Muestreo lógico. En esta técnica se establece un orden ancestral de los nodos de la red, y será el que rija el orden de muestreo de cada componente del individuo que se muestrea. Cada componente i se muestrea según la distribución condicionada de la variable X_i ; esto es $P(X_i|X_{F_i})$. Así, el orden ancestral dice que cuando una variable vaya a ser muestreada previamente ha tenido que muestrear las variables padres.

En este caso, la función de muestreo es la misma distribución de la red bayesiana por lo que el peso de cada individuo adopta la forma, usando la expresión (6.18) en la Ecuación (6.20):

$$\begin{aligned}
 w_t &= \frac{\left[\prod_{i=1}^n P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i}) \right] \left[\prod_{j \in E} \delta_{e_j}(x^{(t)\downarrow j}) \right]}{\prod_{i=1}^n P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i})} \\
 &= \prod_{j \in E} \delta_{e_j}(x^{(t)\downarrow j}) \\
 &= \begin{cases} 1 & \text{si } x^{(t)\downarrow j} = e_j \text{ para todo } j \in E \\ 0 & \text{en otro caso} \end{cases}
 \end{aligned}$$

En este método todos los pesos valen 0 ó 1 dependiendo de si la configuración obtenida es coherente con las observaciones o no. Es más, si durante el muestreo de cada componente se obtiene un valor simulado para un nodo evidencia que no coincide con el valor muestreado se rechaza la muestra completa (peso 0). Si todos los valores simulados para todos los nodos evidencia coinciden con el valor muestreado entonces la muestra se acepta (peso 1). Las probabilidades se aproximan entonces calculando el cociente entre los casos en los que la muestra está en concordancia con la evidencia y el número de muestras obtenido. El algoritmo se muestra en el Algoritmo 6.10.

Algoritmo 6.10 Muestreo lógico.

```

1: {Desarrollo de los pasos 1: a 4: del Algoritmo 6.9.}
2: para  $t = 1, 2, \dots, M$  hacer
3:   para  $i = 1, 2, \dots, n$  hacer
4:     Simular la componente  $x_i$  de  $x^{(t)}$  a partir de  $P(x_i|x_{F_i})$ .
5:     Si  $i \in E$  y  $x_i \neq e_i$ , repetir el ciclo i. {Es asignar peso nulo ( $w_t = 0$ )}
6:   fin para
7:    $w_t = 1$ 
8: fin para

```

A este método también se le conoce como **método de aceptación-rechazo**. Realmente la técnica de método de aceptación-rechazo es una técnica bien conocida en el campo de la simulación. Fue [Henrion, 1988] el primero en aplicarlo en el campo de las redes bayesianas pero lo llamó muestreo lógico.

Ponderación de la Verosimilitud. Surgió con el objetivo de evitar individuos inconsistentes con la evidencia. La propuesta, desarrollada independientemente por [Peot, 1990] y [Chang, 1990], es similar a la de Henrion pero, en vez de simular la evidencia, se toma directamente el valor observado en los nodos evidencia. La distribución de muestreo será por tanto igual que las distribuciones de la red para las variables no observadas o a una distribución degenerada para las variables evidencia:

$$P^*(x) = \prod_{i=1} P^*(x_i) \quad \text{donde} \quad P^*(x_i) = \begin{cases} P(x_i|x_{F_i}) & \text{si } i \notin E \\ \delta_{e_i}(x_i) & \text{si } i \in E \end{cases}$$

El peso de cada individuo adopta ahora la forma:

$$\begin{aligned} w_t &= \frac{\left(\prod_{i=1}^n P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i})\right) \left(\prod_{j \in E} \delta_{e_j}(x^{(t)\downarrow j})\right)}{\left(\prod_{i \notin E}^n P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i})\right) \left(\prod_{j \in E} \delta_{e_j}(x^{(t)\downarrow j})\right)} \\ &= \prod_{i \in E} P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i}) \end{aligned}$$

El peso corresponde a la probabilidad de la evidencia dado el resto de las variables o, equivalentemente, la verosimilitud de la evidencia (de aquí el nombre de este método). Este método, descrito en el Algoritmo 6.11, produce muestras representativas siempre que la probabilidad de la evidencia observada no esté muy próxima a cero porque es posible que entonces gran parte de los datos simulados tengan peso nulo [Hernández y otros, 1998].

Algoritmo 6.11 Ponderación de la verosimilitud.

```

1: {Desarrollo de los pasos 1: a 4: del Algoritmo 6.9.}
2: para  $t = 1, 2, \dots, M$  hacer
3:   para  $i = 1, 2, \dots, n$  hacer
4:     si  $i \notin E$  entonces
5:       Simular la componente  $x_i$  de  $x^{(t)}$  a partir de  $P(x_i|x_{F_i})$ .
6:     fin si
7:     si  $i \in E$  entonces
8:       Asignar a la componente  $x_i$  de  $x^{(t)}$  el valor  $e_i$ .
9:     fin si
10:    fin para
11:    $w_t = \prod_{i \in E} P(x^{(t)\downarrow i} | x^{(t)\downarrow F_i})$ 
12: fin para

```

Simulación estocástica. Es un método propuesto por [Pearl, 1987] y se basa en utilizar como distribución de muestreo la distribución condicionada de cada variable a su envolvente de Markov⁷. El Algoritmo 6.12 detalla su funcionamiento para la estimación de la probabilidad de una variable. El algoritmo se fundamenta en la teoría de los procesos de Markov por lo que la convergencia de la estimación al auténtico valor está asegurada, siempre que la distribución conjunta de la red sea estrictamente positiva. Su convergencia puede verse ralentizada por el hecho de que cada individuo depende del previamente muestreado pudiendo, en el peor de los casos, generarse una configuración con una probabilidad extrema (próxima al 0 o al 1). Esto puede provocar que sea muy difícil obtener configuraciones distintas a las generadas previamente. También requiere de una configuración de partida que involucre a todas las variables y con probabilidad positiva. Dicha configuración se puede obtener aplicando técnicas de muestreo hacia adelante [Jensen y otros, 1995].

6.4.2 Interpretación de evidencias

Con la inferencia probabilística ha aprendido a realizar razonamiento deductivo: derivar consecuencias a partir de un conjunto evidencial. Pero éste no siempre es el *tipo de razonamiento* que se requiere en todas las situaciones. En otras ocasiones se está interesado en realizar razonamiento abductivo: buscar la mejor explicación que justifique la presencia de la evidencia. Por ejemplo, en el campo de la salud, los médicos pueden plantearse un razonamiento consistente en establecer cuál es el conjunto de enfermedades más probables que producen una serie de síntomas en un paciente. La misma situación se presenta cuando en su taller el mecánico debe determinar las causa más probables que hace que su coche no funcione. Estos razonamientos abductivos se suelen presentar en problemas de diagnóstico.

En este contexto, el conjunto de variables observadas X_E , sobre las que se tiene ciertas evidencias e , recibe el nombre de **conjunto manifestación**. La configuración de valores x_I de las variables X_I que maximiza $P(x_I|e)$ recibe el nombre de **explicación más probable** para e y las variables X_I se llama **conjunto de explicaciones**.

⁷La *envolvente (o manto) de Markov* de un nodo en una red bayesiana es el conjunto de nodos formados por sus padres, sus hijos y los padres de sus hijos.

Algoritmo 6.12 Simulación estocástica.

Entradas: Evidencias específicas e del conjunto de variables X_E . Las probabilidades condicionadas de una red bayesiana. El tamaño de la muestra M .

Resultado: Una estimación de la probabilidad condicionada $P(X_i|X_E = e)$ para cada variable de la red.

- 1: **para** $i \in E$ **hacer**
- 2: Definir $x_i = e_i$
- 3: **fin para**
- 4: Establecer $x^{(0)}$ tal que $P(x^{(0)}) > 0$. $\{x^{(0)\perp E} = e\}$
- 5: **para** $i \notin E$ **hacer**
- 6: Definir $w_i(x_i) = 0$
- 7: **fin para**
- 8: **para** $t = 1, 2, \dots, M$ **hacer**
- 9: **para** $i = 1, 2, \dots, n$ **hacer**
- 10: Calcular los valores $P(x_i|x_{M_i})$ según la expresión

$$P(x_i|x_{M_i}) = \alpha P(x_i|x_{F_i}) \prod_{j \in H_i} P(x_j|x_{F_j})$$

donde α es una constante de normalización, x_{M_i} representa la configuración actual de la envolvente de Markov de X_i y H_i representa al conjunto de hijos de i .

- 11: Simular un valor $x^{(t)\perp i}$ a partir de la distribución $P(x_i|x_{M_i})$
- 12: Utilizar alguna de las siguientes expresiones para actualizar w_i :

$$\begin{aligned} w_i(x^{(t)\perp i}) &:= w_i(x^{(t)\perp i}) + 1 \\ w_i(x^{(t)\perp i}) &:= w_i(x^{(t)\perp i}) + P(x^{(t)\perp i}|x_{M_i}). \end{aligned}$$

- 13: **fin para**
 - 14: **fin para**
 - 15: **para** $i = 1, 2, \dots, n$ **hacer**
 - 16: Definir, para cada x_i , $\hat{P}(x_i|e) = w_i(x_i) / \sum w_i(x_i)$ {Normalizar los valores w_i }.
 - 17: **fin para**
 - 18: **devolver** $\{\hat{P}(X_i|e)|i = 1, \dots, n\}$ {Como el auténtico valor de cada $P(X_i|e)$.}
-

Si I abarca todas las variables de la red se habla de **abducción total** o simplemente abducción. Cuando se quiere sólo el estado más probable de unas pocas variables de la red se habla de **abducción parcial**. En ambos casos puede estar interesado no sólo en la explicación más probable, sino en cuáles son las K *primeras* configuraciones que maximizan $P(x_I|e)$, en cuyo caso se habla del problema de las K **configuraciones más probables**. El problema de encontrar las K configuraciones completas más probables en una red bayesiana es un problema de optimización referenciado también como el problema de la revisión de creencias [Pearl, 1988], búsqueda de la configuración máxima a posteriori [Shimony y Charniak, 1990] o búsqueda de la explicación más probable [Sy, 1993].

Dada una red, uno podría pensar que la configuración buscada x^* pudiera obtenerse a partir de las instancias x_i^* de las variables que maximizan cada una de las probabilidades condicionadas. Desgraciadamente, si $x^* = \arg \max_x P(x_N|e)$ y $x_i^* = \arg \max_{x_i} P(x_i|x_{F_i})$ lo que suele producirse es que $x^* \neq (x_1^*, x_2^*, \dots, x_n^*)$. El problema es bastante más complejo, desarrollándose distintas aproximaciones para resolverlo. Entre las pioneras destacan la de [Cooper, 1984], que desarrolla un algo-

ritmo basado en primero el mejor con poda. [Pearl, 1988] describe un algoritmo que encuentra la configuración más probable en políárboles. [Sy, 1993] se basa en este trabajo para encontrar las K configuraciones más probables para ese tipo de redes. Estas técnicas se basan en la idea de mandar ciertos mensajes informativos entre los nodos de forma similar al mostrado en la Figura 6.7. El problema está en su extensión a grafos dirigidos acíclicos en general. Al igual que en el problema de la inferencia probabilística no es posible mantener la estructura original y algunos algoritmos requieren de su manipulación para construir nuevas estructuras arbóreas, donde cada nodo está formado por varias variables de la red original para posteriormente mandar mensajes informativos entre nodos, como en [Dawid, 1992] o [Seroussi y Goldmard, 1994].

Básicamente, el planteamiento y soluciones para calcular la explicación más probable son similares al problema de la inferencia probabilística, aunque tiene, como es de esperar, sus propios matices. La diferencia sustancial es que en vez de marginalizar mediante sumas debe de hacerlo tomando el máximo. En efecto, de forma análoga a la propiedad distributiva de la suma para la inferencia, dispone ahora de la siguiente propiedad distributiva para la función máximo $\max\{xy, xz\} = x \max\{y, z\}$ que, expresado en términos de funciones, establece: “**Si x no está en el dominio de la función f , entonces $\max_x(f \times g) = f \times \max_x g$ para cualquier otra función g** ”.

Por ejemplo, sobre una probabilidad conjunta de tres variables se obtienen las siguientes expresiones:

$$x^* = (x_1^*, x_2^*, x_3^*) = \arg \max_{x_1, x_2, x_3} P(x_1, x_2, x_3) \quad (6.21)$$

$$= \arg \max_{x_1, x_2, x_3} P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \quad (6.22)$$

$$= \arg \max_{x_1} \max_{x_2} \max_{x_3} P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \quad (6.23)$$

$$= \arg \max_{x_1} P(x_1) \left\{ \max_{x_2} P(x_2|x_1) \left[\max_{x_3} P(x_3|x_1, x_2) \right] \right\} \quad (6.24)$$

La última expresión nos indica que la configuración buscada se puede obtener realizando un recorrido de derecha a izquierda consistente en los siguientes pasos reiterativos, hasta conseguir la configuración x^* :

1. Instanciar parcialmente algunas distribuciones de la red. Las instancias parciales corresponden a las configuraciones que maximizan las combinaciones situadas más a la derecha.
2. Desarrollar combinaciones que involucren a algunas de esas distribuciones parcialmente instanciadas.
3. Posteriormente, tomar configuraciones que maximizan esas combinaciones.

Esta idea, es similar a la vista en inferencia probabilística (véase la Ecuación 6.13). De nuevo, es interesante observar que todas las funciones que no mencionan a una variable X_i pueden desplazarse a la izquierda de la expresión a maximizar, ya que

respecto a X_i sus valores actúan como constantes. Por ejemplo, para una distribución de tres variables con la factorización anterior sólo se requiere las funciones $P(x_2|x_1)$ y $P(x_3|x_1, x_2)$ si está interesado en $\max_{x_2, x_3} P(\cdot)$ o $\max_{x_2} P(\cdot)$; y sólo se necesita $P(x_3|x_1, x_2)$ para obtener el valor x_3^* que maximice la distribución conjunta $P(\cdot)$.

Algoritmo 6.13 Eliminación por máximo de la variable X_i de un conjunto H de funciones.

Entradas: Una variable, X_i . Un conjunto de funciones H , algunas definidas sobre X_i .

Resultado: La configuración x_i^* que maximiza las funciones de H . Un conjunto de funciones que no contienen a X_i .

- 1: Seleccionar aquellas funciones h de H que contienen a la variable X_i de interés. Llamar H_i al conjunto de dichas funciones.
 - 2: Calcular f_i como el producto de todas las funciones de H_i .
 - 3: Determinar $x_i^* = \arg \max_{x_i} f_i$
 - 4: Redefinir f_i como el resultado de instanciar la función al valor x_i^* ; $f_i := \max_{x_i^*} \prod_{f \in H_i} f$.
 - 5: Redefinir $H := [H \setminus H_i] \cup \{f_i\}$.
 - 6: **devolver** (x_i^*, H)
-

Si por **eliminar una variable por máximos** se entiende a la acción de instanciar al valor que maximiza la expresión que se encuentra a su derecha, la solución al problema de la explicación más probable puede contemplarse como un proceso consistente en la eliminación de los nodos de la red según cierto orden establecido. Esta idea es la que se muestra en el Algoritmo 6.14. En primer lugar se reduce la dimensionalidad de las matrices informativas, al igual que se hizo en el Algoritmo 6.7.

A continuación, se recorren todas las variables de la red y se comprueban si son variables observadas o no. Si una variable lo es, su configuración más probable viene dada (es la evidencia). Si la variable no es una evidencia, se elimina por máximos (Algoritmo 6.13). Primero se descartan las funciones de la red que actúan como constantes en el proceso de maximización de la variable (las que se desplazarían al lado izquierdo). Después, se obtiene la instancia de la variable que maximiza la combinación de las funciones que la contiene y se restringen parcialmente los valores de la funciones a la configuración obtenida. Al final del proceso de eliminación se obtiene la asignación más probable.

Para la tarea de determinar la máxima hipótesis a posteriori (MAP), la pregunta se formula sobre la máxima creencia asociada a un subconjunto de variables hipótesis no observadas:

$$P(x_I^*|e) = \max_{x_I} P(x_I|e) \propto \max_{x_I} P(x_i, x_E = e) = \max_{x_I} \sum_{x_{N \setminus I \cup E}} \left[\prod_{i=1}^n P(x_i|x_{F_i}) \right]_{|x_E=e}$$

Esta tarea es una generalización tanto de MPE como de asignación de creencias a posteriori; de hecho, es, por definición, una mixtura de los dos problemas de inferencia que se han estudiado hasta ahora. Así, algunas variables serán eliminadas por sumas y otras por máximos. Para que la ordenación sea más efectiva la secuencia de eliminación por sumas y máximos deberá permitir que se intercalen de forma adecuada (véase el Ejercicio 6.15).

Algoritmo 6.14 Cálculo de la explicación más probable en una red bayesiana.

Entradas: Las probabilidades condicionadas de una red bayesiana. Evidencias específicas e del conjunto de variables X_E .

Resultado: Determinación de la MPE.

```

1: para cada  $l \in E$  hacer
2:   Definir  $P'(X_i|X_{F_i}) := P(X_i|X_{F_i}) \prod_{j \in E} \delta_{e_j}(x_j)$  para cada función  $P(X_i|X_{F_i})$  tal que  $l \in \{i\} \cup F_i$  y redefinir  $H := H \cup \{P'(X_i|X_{F_i})\} \setminus \{P(X_i|X_{F_i})\}$ .
3: fin para
4: para cada  $i \in N$  hacer
5:   si  $i \in E$  entonces
6:     Determinar  $x_i^* := e_i$ .
7:   fin si
8:   si  $i \notin E$  entonces
9:      $(x_i^*, H) = \text{Algoritmo-6.13}(X_i, H)$ 
10:  fin si
11: fin para
12: devolver  $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ 
```

6.5 Referencias bibliográficas y software

En este capítulo se le ha mostrado una introducción al cálculo de probabilidades y las redes bayesianas. Algunos resultados se han comentado muy superficialmente y otros muchos ni tan siquiera se han mencionado. Hay muchos buenos libros introductorios de la teoría de la probabilidad. La visión de [DeGroot, 1989] combina probabilidad y estadística de un modo sencillo y desde un punto de vista bayesiano. Para redes bayesianas algunos de los libros monográficos sobre este tema y que le puede ayudar en su estudio son:

- Probabilistic Reasoning in Intelligence Systems [Pearl, 1988]. Sin duda la publicación más importante sobre el tema y que ha hecho que las redes bayesianas hallan experimentado tan rápido desarrollo.
- Graphical Belief Modelling [Almogn, 1995].
- Expert Systems and Probabilistic Network Models [Castillo y otros, 1997].
- Probabilistic Networks and Expert Systems [Cowell y otros, 1999].
- Learning Bayesian Network [Neapolitan, 1990].
- Bayesian Networks and Decision Graphs [Jensen, 2001].
- Bayesian Artificial Intelligence [Korb y Nicholson, 2004].

La investigación en este tema se produce de manera continua y sus investigadores se suelen dar cita en dos grandes congresos internacionales relevantes sobre redes bayesianas:

- *Conference on Uncertainty in Artificial Intelligence* (UAI). Es un conferencia americana anual y foro principal para investigadores que trabajan en redes bayesianas y estructuras afines como redes de Markov, grafos cadena, diagramas de influencia y en general sobre cualquier modelo gráfico.

- *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (SQARU). Es la contrapartida europea que se celebra cada dos años. Es un foro más amplio destinado a cualquier aproximación para el cálculo de la incertidumbre. Los resultados se publican en *Lectures Notes in Artificial Intelligence* de Springer-Verlag.

Nuevas investigaciones sobre modelos gráficos aparecen en revistas sobre inteligencia artificial como *Artificial Intelligence* o *Journal of Artificial Intelligence Research*. Pero muchos otros aparecen en revistas más relacionadas con el campo estadístico.

En la actualidad, ya hay muchos sistemas software para trabajar con redes bayesianas e incluso algunos paquetes estadísticos conocidos han aumentado sus módulos para contemplar el tratamiento de datos con estas estructuras. En la Tabla 6.1 se muestra un listado de software con su dirección de descarga y que puede obtenerse libre de carga, al menos a nivel académico, e incluso alguno pone las API y/o el código fuente a disposición de los usuarios como es el caso de *Elvira*, resultado de la cooperación de cinco grupos de investigación de distintas universidades españolas.

Nombre	Url
Elvira	leo.ugr.es/elvira
Hugin	www.hugin.com
Bayesware	www.bayesware.com
BN Toolbox	www.cs.berkeley.edu/~murphyk/Bayes/bnsoft.html
BucketElim	www.ice.uci.edu/~irinar
Genie	www2.sis.pitt.edu/~genie
Java Bayes	www.cs.cmu.edu/~javabayes/Home
Netica	www.norsys.com
XBAIES	www.staff.city.ac.uk/~rge/webpages/xbpage.html

Tabla 6.1: Software para el tratamiento de redes bayesianas.

6.6 Ejercicios propuestos

6.1. En la interpretación clásica las probabilidades se asignan proporcionales a las verosimilitudes. ¿Bajo qué circunstancias tiene esto sentido?

6.2. Considerar un dado de seis caras y enumeradas del 1 al 6. ¿Cuál es la probabilidad de que salga un 6? ¿y de que salgan dos 6? ¿y de que salga un 6, dado que en el último lanzamiento salió un 6? Estudiar su cálculo según las distintas interpretaciones de probabilidad. ¿Cuál debería ser su asignación si el dado está trucado?

6.3. Dada una baraja de 52 cartas se consideran los siguientes atributos con sus correspondiente valores:

- *Número*: Toma el mismo valor de la carta ($1, 2, \dots, Q, K$).
- *Figura*: Booleana. Ciento si la carta es J, Q o K .

- *Reina*: Booloeana. Ciento si la carta es *Q*.
- *Palo*: Toma valores en {*Trebol, Pica, Corazón, Diamante*}.
- *Color*: Puede ser *Rojo* o *Negro*.

Si se extraen (con reemplazamiento) cartas al azar, compruebe qué tipo de independencias existen entre estas variables dos a dos, tres a tres, etc. Por ejemplo ¿el *Palo* es independiente del *Número*? ¿el *Color* es independiente de ser *Reina*, conocido el *Palo*? ¿es el *Número* independiente de {*Reina, Palo*} dado el *Color*?, etc

6.4. Imaginar el espacio probabilístico $U = \{\text{personas de una ciudad}\}$ formado por n personas y considerar las variables booleanas $X = \{\text{tener ojos azules}\}$ e $Y = \{\text{tener el cabello negro}\}$ con valores $\{v = \text{Verdad}, f = \text{Falso}\}$. Puede determinar las probabilidades $P(X)$, $P(Y)$ y $P(X, Y)$ asignando, como tales, los valores obtenidos por sus frecuencias relativas (¿sabe qué interpretación se está aplicando?):

$$P(X = v) = \frac{f_X}{n}; \quad P(Y = v) = \frac{f_Y}{n}; \quad P(X = v, Y = v) = \frac{f_{X \wedge Y}}{n}$$

donde f_T representa al número de personas para los que T es verdad. Si al pasear por la calle ve a una persona por detrás con el cabello negro ¿cuál debería ser su creencia de que además tenga los ojos azules? (sin mirarle la cara, claro). Esta probabilidad, a determinar, es $P(X = v|Y = c)$. Calcular y comprobar que obtiene precisamente la expresión (6.3) de la Definición 6.6.

Demuestre la Ecuación (6.3).

6.5. Después de una sesión de conexión a internet, el ordenador da positivo en una prueba antivirus por la presencia de un tipo concreto de virus. El tipo de prueba realizada acierta en un 98 % si el sistema contiene este tipo de virus. Un falso positivo para ese virus se establece en un 5 %. Si como usuarios sabemos que el sistema se ha infectado en un M % de las conexiones ¿cuál es la probabilidad a posteriori de que el sistema esté infectado habiendo dado positivo el antivirus?

6.6. Construir una red bayesiana que refleje esta situación. Imagine que instala en su casa una alarma para evitar posibles robos. Su alarma no es perfecta y, en ocasiones, también puede dispararse por temblores en el suelo: por ejemplo, por culpa de un terremoto. Su vecino Juan es asustadizo, confunde su alarma con cualquier sonido similar y le llama constantemente a su trabajo en cuanto cree que su alarma se activa. Por contra su vecina María es arisca y sólo le avisará si realmente considera que le están robando. Si está en su trabajo, le llaman porque suena su alarma y de camino a su casa se entera por la radio de que se ha producido un terremoto ¿cómo debería modificar su credibilidad en que le están robando? Estudie cuidadosamente las dependencias normales e inducidas. Asigne probabilidades coherentes a la situación.

6.7. Justificar mediante argumentos semánticos por qué una red causal debe ser acíclica. Justificar mediante argumentos matemáticos (resultados probabilísticos) por qué una red bayesiana debe ser acíclica.

6.8. A partir de tres variables construir cada uno los posibles grafos dirigidos que pueden formarse con 0 arcos, 1 arcos, 2 arcos y 3 arcos. Construir la factorización de la función de probabilidad sugerida por cada uno de los grafos que haya construido.

6.9. La regla de Bayes puede utilizarse para resolver problemas de clasificación. En estos problemas se identifica cada individuo por una instancia de n -valores correspondientes a n -variables, $\{X_1, X_2, \dots, X_n\}$. El objetivo es indicar, a partir de los valores (x_1, x_2, \dots, x_n) , la clase (categoría o grupo) más probable a la que pertenece el individuo. Para ello se considera una variable distinguida, C , que tiene tantos casos como clases se consideren.

El modelo más sencillo es el conocido como Naive Bayes que establece las siguientes reglas:

- Cada clase de C es responsable los valores que toma cada variable X_i .
- Un atributo X_i es independiente de los demás, conocida la clase a la que pertenece el individuo.

¿Cuál debería ser la red bayesiana asociada a este modelo? ¿Qué tipo de razonamiento debe aplicarse? Indique analíticamente la expresión matemática que debe calcular.

6.10. Un experto no es capaz de suministrarle una probabilidad conjunta sobre seis variables, pero sí es capaz de suministrarle las siguientes probabilidades condicionadas:

$$\{P(x_1|x_2, x_3), P(x_2|x_4), P(x_3|x_4), P(x_4), P(x_5|x_2), P(x_6|x_3)\}$$

1. Determinar un grafo G para el cual P cumpla la condición de Markov.
2. ¿Cuántas ordenaciones para la descomposición de la regla de la cadena existen para este modelo probabilístico?
3. ¿Cuántas redes bayesianas puede construir para las probabilidades dadas?

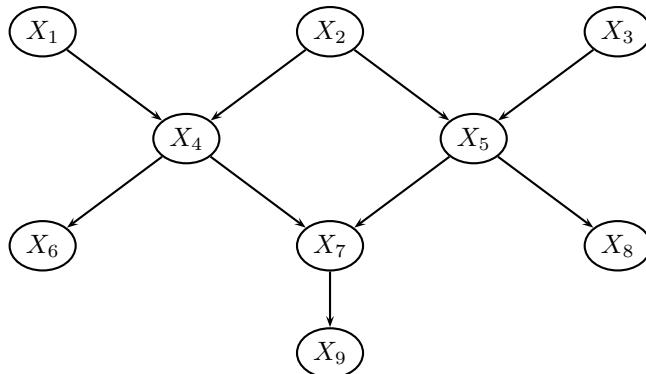


Figura 6.10: Una red bayesiana.

6.11. A partir de la red bayesiana de la Figura 6.10, determinar para cada par de variables no adyacentes, X e Y , los conjuntos minimales de variables, K_I , que las hace independientes; es decir, $I(X, Y|X_I)$. Hacer lo mismo para los conjuntos minimales de variables, K_D , que las hace dependientes; es decir, $\neg I(X, Y|X_D)$.

6.12. Si en la red bayesiana de la Figura 6.10, se considera que la variable X_i tienen c_i -casos, con $c_i = \max\{2, i \bmod 6\}$, ¿cuáles son las secuencias de eliminación óptimas para calcular $P(X_6|e_3)$, $P(X_8, X_1|e_3)$ y la MPE, respectivamente?

6.13. Para la red bayesiana de la Figura 6.10, considerar que cada variable está formada por dos casos, $\{0, 1\}$. Se considera que las variables X_4 y X_9 están instanciadas al valor 0. Asignar probabilidades cualesquiera a las variables, pero establecer las siguientes a los nodos X_4 y X_9 :

X_1	X_2	$X_4 = 0$
0	0	c
0	1	c
1	0	c
1	1	c

X_7	$X_9 = 0$
0	c
1	c

$$\text{y} \quad c = 0,3$$

- Utilizar cualquier herramienta de la Tabla 6.1 y calcular las probabilidades a posteriori de las variables no observadas.
- Calcular dichas probabilidades utilizando la técnica de eliminación por sumas.
- Determinar todas las ordenaciones ancestrales para la Figura 6.10. Buscar una ordenación eficiente que rechace realizaciones incongruentes con la evidencia.
- Desarrollar el algoritmo del muestreo lógico para muestras de tamaño 10, 50 y 100 respectivamente.
- Analizar la convergencia de las probabilidades aproximadas para las tres tamaños para valores $c = 0,3$, $c = 0,01$, $c = 0,00001$.
- ¿Cuál es el porcentaje de individuos rechazados para cada uno de los valores c ?
- Comparar la convergencia del muestreo lógico con la de la ponderación de la verosimilitud y la simulación estocástica.

6.14. Analizar la frase: *El Algoritmo 6.11, produce muestras representativas siempre que la probabilidad de la evidencia observada no esté muy próxima a cero porque es posible que entonces gran parte de los datos simulados tengan peso nulo.* Poner un ejemplo de cuándo no se obtienen muestras representativas.

6.15. Diseñar un algoritmo para obtener la MAP utilizando los algoritmos de eliminación por sumas y eliminación por máximos. Tomar como referencia los siguientes cálculos sobre una conexión en serie $X_i \rightarrow X_{i+1}$ ($i = 1, 2, 3$), con X_3 observada al valor e_3 :

$$\max_{x_1, x_3} P(x_1, e_3) = \max_{x_1, e_3} \sum_{x_2, x_4} P(x_1)P(x_2|x_1)P(e_3|x_2)P(x_4|x_3)$$

$$\max_{x_1, x_4} P(x_1, x_4) = \max_{x_1, x_4} \sum_{x_2, x_3} P(x_1)P(x_2|x_1)P(e_3|x_2)P(x_4|e_3)$$

Determinar empíricamente cómo deberían intercalarse óptimamente las eliminaciones por sumatorias y por máximos.

Referencias

- ALMONG, RUSELL: *Graphical Belief Modelling*. Chapman & Hall, 1995.
- CANO, A. y MORAL, S.: «Heuristic algorithms for the triangulation of graphs.» *Advances in Intelligent Computing*, 1995, pp. 166–171.
- CASTILLO, ENRIQUE; GUTIÉRREZ, JOSÉ M. y HADI, ALI S.: «Sistemas Expertos y Modelos de Redes Probabilísticas». En: *Monografías de la Academia de Ingeniería, Academia de Ingeniería*, 1996.
- CASTILLO, ENRIQUE; GUTIÉRREZ, JOSÉ M. y HADI, ALI S.: *Expert Systems and Probabilistic Network Models*. Springer-Verlang, 1997.
- CHANG, ROBERT FUNG & KUO-CHU: «Weighing and integrating evidence for stochastic simulation in bayesian networks». En: M.Henrion & R.D.Shachter & L.N.Kanal & J.F.Lemmer (Ed.), *Uncertainty in artificial intelligence*, 5, pp. 209–219. Elsevier science publisher B.V. (North-Holland), 1990.
- COOPER, G.F.: «NESTOR: a computer-cased medical diagnostic that integrates causal and probabilistic knoledge». *Informe técnico*, Stanford University. California, 1984.
- COOPER, G.F.: «The Computational complexity of probabilistic inference using Bayesian belief networks». *Artificial Intelligence*, 1990, (42), pp. 393–405.
- COWELL, ROBERET G.; DAWID, A. PHILIP y LAURITZEN, STEFFEN L.: *Probabilistic Networks and Expert Systems*. Springer-Verlang, 1999.
- DAGUM, P. y LUBY, M.: «Approximating probabilistic inference in Bayesian netwroks in NP-hard». *Artificial Intelligence*, 1993, (60), pp. 141–153.
- DAWID, A. P.: «Applications of a general propagation algorithm for probabilistic expert systems». *Statistics and Computing*, 1992, (2), pp. 25–36.
- DECHTER, R.: «Bucket elimination: A unifying framework for reasoning». *Artificial Intelligence*, 1999, (113), pp. 41–85.
- DEGROOT, M. H.: *Probability and Statistics*. Addison-Wesley, 1989.
- HENRION, M.: «Propagation of Uncertainty by Probabilistic Logic Sampling in Bayes Networks». En: J. Lemmer & L.N. Kanal (Ed.), *Uncertainty in artificial intelligence*, 2, pp. 149–164. Elsevier science publisher B.V. (North-Holland), 1988.
- HERNÁNDEZ, L.D.; MORAL, S. y SALMERÓN, A.: «A Monte Carlo algorithm for probabilistic propagation in belief networks based on importance sampling and stratified simulation techniques». *International Journal of Appoximate Reasoning*, 1998, (3), pp. 53–91.

- HUANG, C. y DARWICHE, A.: «Inference in belief networks: A procedural guide». *International Journal of Approximate Reasoning*, 1996, (15), pp. 225–263.
- JENSEN, C.S.; KONG, A. y KJÆRULFF, U.: «Blocking Gibbs Sampling in Very Large Probabilistic Expert Systems». *International Journal of Human-Computer Studies*, 1995, (42), pp. 647–666.
- JENSEN, FINN V.: *An Introduction to Bayesian Network*. UCL Press London, 1996.
- JENSEN, FINN V.: «Bayesian Networks and Decision Graph». En: *Statistics for Engineering and Information Science*, Springer, 2001.
- JENSEN, F.V.; LAURITZEN, S.L. y OLESEN, K.G.: «Bayesian updating in causal probabilistic networks by local computations». *SIAM Journal on Computing*, 1990, (4), pp. 269–282.
- KEYNES, J.M.: *A Treatise on Probability*. Macmillan, London, 1921.
- KOLMOGOROV, A.N.: *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer, Berlin, 1933.
- KORB, KEVIN B. y NICHOLSON, ANN E.: *Bayesian Artificial Intelligence*. Chapman & Hall/CRC, 2004.
- LAURITZEN, S. y SPIEGELHALTER, D.J.: «Local computations with probabilities on graphical structures and their application to expert systems (with discussion)». *Journal of the Royal Statistical Society Series B*, 1988, (50), pp. 157–224.
- LAW, W.D., A.M. & KELTON: *Simulation Modeling & Analysis*. McGraw-Hill, 1991.
- LI, Z. y D'AMBROSIO, B.: «Efficient inference in Bayes nets as a combinatorial optimization problem». *International Journal of Approximate Reasoning*, 1994, (11), pp. 55–81.
- MARTIN, A. TANNER: *Tools for Statistical Inference. Methods for the exploration of posterior distributions and likelihood functions*. Springer Series in Statistics, 1996.
- NEAPOLITAN, R. E.: *Probabilistic Reasoning in Expert Systems*. Wiley and Sons, New York, 1990.
- PEARL, J.: «Evidential reasoning using stochastic simulation of causal models». *Artificial Intelligence*, 1987, (32), pp. 247–257.
- PEARL, JUDEA: *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA., 1988.
- PEOT, R.D. SHACHTER & M.A.: «Simulation approaches to general probabilistic inference on belief networks». En: M.Henrion & R.D.Shachter & L.N.Kanal & J.F.Lemmer (Ed.), *Uncertainty in artificial intelligence*, 5, pp. 221–231. Elsevier science publisher B.V. (North-Holland), 1990.

- RUBINSTEIN, R.Y.: *Simulation and the Monte Carlo Method*. John Wiley & Sons, 1981.
- SEROUSSI, B. y GOLDMARD, J.L.: «An algorithm directly finding the k most probable configurations in bayesian networks». *International Journal of Approximate Reasoning*, 1994, (11), pp. 205–233.
- SHACHTER, R.D.; ANDERSEN, S.K. y SZLOVITS, P.: «The equivalence of exact methods for probabilistic inference on belief network». *Artificial Intelligence*, 1991.
- SHENOY, P.P.: «A valuation-based language for expert systems». *International Journal of Approximate Reasoning*, 1989, (5), pp. 383–411.
- SHIMONY, S.E. y CHARNIAK, E.: «A new algorithm for finding map assgnments to bellief networks». En: *Proceedings of the 6th conferencie on Unvertainty in Artificial Intelligence*, , 1990.
- SY, B.K.: «A recurrence local computation approach towards ordering composite beliefs in Bayesian belief networks». *International Journal of Approximate Reasoning*, 1993, (8), pp. 17–50.
- VON MISES, R.: *Probability, Statistics, and Truth*. Allen & Unwin, London, 1957. (Publidado originalmente en 1928).
- WEN, W.X.: «Optimal decomposition of belief networks». *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, 1990, (Cambridge, MA), pp. 245–256.
- ZHANG, N.L. y POOLE, D.: «Exploiting causal independence in Bayesian networkd inference». *Journal of Artificial Intelligence Research*, 1996, (5), pp. 301–328.

Capítulo 7

Conjuntos borrosos

Paulo Félix Lamas y Alberto José Bugarín Diz
Universidad de Santiago de Compostela

7.1 Introducción

Históricamente, las ciencias formales han restringido el uso del lenguaje natural a aquellos predicados que, aplicados a cierta colección de objetos U , dividen ésta en dos subconjuntos. Así, si U es el conjunto de los números naturales $U = \{1, 2, 3, 4, \dots\}$, el predicado ‘impar’ lo divide en dos subconjuntos. Para cada uno de los elementos de uno de ellos: $I = \{1, 3, 5, 7, \dots\}$, decimos que el predicado ‘impar’ es cierto, mientras que para el otro: $N = \{2, 4, 6, 8, \dots\}$, este predicado es falso. El predicado ‘impar’ pertenece a la clase de los *predicados precisos*, también denominados predicados clásicos, por ser el objeto de estudio de la lógica clásica, ciencia de los principios formales y normativos del razonamiento, y que nos permite evaluar la corrección de un discurso.

Tradicionalmente, han quedado excluidos de toda consideración formal otro tipo de predicados, que como ‘alto’, no permiten realizar una división satisfactoria de una población de individuos en dos subconjuntos, aquéllos para los cuales ‘alto’ es cierto, y aquéllos para los que es falso. Y sin embargo, estos predicados, que llamaremos *predicados vagos*, protagonizan el lenguaje ordinario, lo dotan de flexibilidad y constituyen un elemento fundamental en la capacidad de abstracción del ser humano. Sentencias del tipo de ‘velocidad moderada’, ‘bajas presiones’, ‘incremento súbito’ o ‘amor eterno’ aparecen con asiduidad en la toma de decisiones, y son una herramienta imprescindible en la resolución eficiente de problemas. Por contra, la falta de nitidez de estos predicados supone una fuente de inconsistencias; pensemos sin ir más lejos que un individuo puede ser al mismo tiempo alto, y no alto.

La publicación del trabajo de Lotfi A. Zadeh “*Fuzzy Sets*” [Zadeh, 1965] (Conjuntos Borrosos) en 1965 supone un salto cualitativo en la relación entre la ciencia y el lenguaje. Si tradicionalmente el pensamiento científico occidental ha mantenido una relación difícil con el lenguaje ordinario, “*Fuzzy Sets*” plantea la necesidad de utilizar herramientas formales para su análisis, y así surge la Teoría de los Conjuntos Borrosos, como la teoría que ha de proporcionar dichas herramientas, y la Lógica Borrosa, como la ciencia de los principios formales del razonamiento aproximado.

7.2 Conjuntos borrosos

La noción de predicado introducida en la sección anterior está íntimamente ligada a la de conjunto, de modo que todos aquellos números naturales que verifican el predicado ‘impar’ pertenecen al conjunto I . En general, todo predicado preciso P , aplicado a una colección U , tiene asociado un conjunto preciso que denotaremos de la misma forma $P \subset U$, y que puede describirse mediante una función $\mu_P(u)$, cuyo valor viene determinado por la pertenencia a P de los distintos elementos $u \in U$, y definida como:

$$\mu_P(u) = \begin{cases} 0 & \text{si } u \notin P \\ 1 & \text{si } u \in P \end{cases}$$

Se dice que para aquellos u para los que $\mu_P(u) = 0$ el predicado P es falso, y para aquellos u para los que $\mu_P(u) = 1$ el predicado P es verdadero.

En cambio, un predicado vago V , aplicado a la misma colección U , tiene asociado un *conjunto borroso* que denotaremos de la misma forma $V \subset U$, y que puede describirse mediante una función $\mu_V(u)$, cuya representación de la pertenencia a V de los distintos elementos de U es una cuestión de grado, de modo que:

$$\mu_V : U \rightarrow [0, 1]$$

Además de aquellos elementos para los que el predicado V es cierto ($\mu_V(u) = 1$), y aquéllos para los que V es falso ($\mu_V(u) = 0$), existe un conjunto de elementos para los que el predicado V es cierto en un determinado grado $0 < \mu_V(u) < 1$.

Así definido, el concepto de conjunto borroso extiende la noción clásica de conjunto. Nada decimos sobre la naturaleza de la función de pertenencia, ni de cómo podríamos determinarla a partir de la experiencia de una comunidad de hablantes, de la experimentación, o del contexto de uso [Black, 1937]. Lo que sí parece evidente es que su forma debe representar adecuadamente el significado que proporcionamos al predicado correspondiente, por lo que cualquiera de las representaciones que vemos en la Figura 7.1 podrían ser ejemplos perfectamente válidos.

Puede resultar claramente cuestionable la asignación de un valor de pertenencia preciso para un elemento determinado $u \in U$. Imaginemos lo verosímil o útil que puede resultar el afirmar que un individuo es alto con grado 0,32. Parece más razonable pensar que la función de pertenencia encuentra su sentido en su capacidad para ordenar los elementos de U según su grado de pertenencia. Por tanto, más que el valor exacto de pertenencia de un individuo al conjunto de los altos, la función de pertenencia proporciona una ordenación del conjunto de individuos que nos permite decir que un individuo “ u es menos alto que w ”.

La elección de la forma de la función de pertenencia es pues una cuestión de diseño. Podemos buscar una representación computacionalmente compacta y eficiente, o una representación con ciertas propiedades matemáticas deseables, tales como la continuidad o la derivabilidad, en función de los requerimientos del problema.

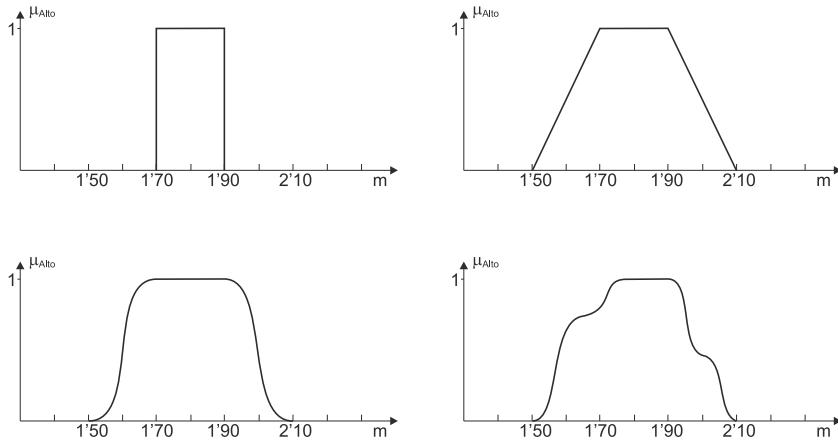


Figura 7.1: Diferentes representaciones de un mismo conjunto borroso, correspondiente al predicado ‘alto’, donde la primera de ellas coincide con una representación clásica.

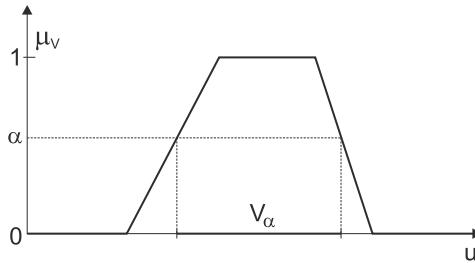
Resulta útil definir los siguientes conjuntos asociados a una función de pertenencia μ_V : su *soporte*, definido como $\{u \in U : \mu_V(u) > 0\}$, y su *núcleo*, definido como $\{u \in U : \mu_V(u) = 1\}$. Aquellos elementos que pertenecen al núcleo de V se consideran *prototipos* de V . Es habitual caracterizar un conjunto borroso mediante las propiedades de convexidad y normalidad. Un conjunto V se dice *convexo* si $\forall u, u', u'' \in U$, $u', u'' \in [u, u'']$, $\mu_V(u') \geq \min\{\mu_V(u), \mu_V(u'')\}$. Un conjunto V se dice *normalizado* si $\exists u \in U$, tal que $\mu_V(u) = 1$.

Una manera interesante de representar un conjunto borroso es a través de una familia de conjuntos precisos anidados, haciendo uso de la noción de α -corte. El α -corte V_α del conjunto borroso V se define como el conjunto $\{u \in U : \mu_V(u) \geq \alpha\}$, para cualquier $0 < \alpha \leq 1$ (ver Figura 7.2). El 1-corte de V coincide con su núcleo. Vemos cómo los miembros de un determinado α -corte son aquellos elementos cuyo grado de pertenencia es mayor o igual que α . Si un conjunto borroso V es convexo, entonces cualquiera de sus α -cortes es un intervalo. El interés de esta representación reside en que podemos descomponer aquellas operaciones que involucren conjuntos borrosos en operaciones con conjuntos precisos, en algunos casos mucho más sencillas.

Dados dos conjuntos borrosos $A, B \subset U$, decimos que “ A está contenido en B ”, y escribimos $A \subseteq B$, si y sólo si, $\forall u \in U$:

$$\mu_A(u) \leq \mu_B(u)$$

Si $A \subseteq B$ y $B \subseteq A$ entonces, obviamente, $A = B$. A lo largo de este capítulo veremos que es posible definir otras medidas de inclusión. Por otra parte, vemos que la noción de función de pertenencia nos permite identificar a un conjunto con una función, con lo que las operaciones conjuntistas pueden entenderse como operaciones entre funciones, lo que estudiaremos en la sección 7.4.

Figura 7.2: Representación del concepto de α -corte.

7.3 Semántica de los conjuntos borrosos

La idea de conjunto es por definición abstracta: una reunión de elementos que comparten alguna propiedad. Esta propiedad es la que determina la pertenencia de un elemento particular al conjunto. Si el conjunto es borroso, la pertenencia es una cuestión de grado. En la sección anterior hemos explicado que la función de pertenencia nos permite dotar de significado a un predicado del lenguaje, nos dice qué es y qué no es alto, y en qué medida. Sin embargo, el significado también depende del uso que se hace de dicho predicado. Y aunque no se conciben límites para el rango de usos en el lenguaje, existen algunos tipos bien reconocibles en la ingeniería, correspondientes a ciertas tareas de interés. Estos tipos definen algunas semánticas que aquí introducimos, sin pretensión de exhaustividad:

- *Similitud:* Sea V un conjunto borroso. Si pensamos en los elementos del núcleo de V como en prototipos de V , entonces $\mu_V(u)$ es el grado de proximidad de u al tipo de elementos prototipo de V . Esta semántica es de particular interés para todas aquellas tareas de clasificación, regresión o agrupamiento, donde se realiza una operación de abstracción a partir de un conjunto de datos mediante alguna medida de proximidad. También es la semántica que caracteriza a los sistemas de control, donde la diferencia, complementaria de la similitud, entre la situación actual y la deseada (prototipo), determina la realización de acciones de corrección.
- *Preferencia:* Supongamos que V reúne a un conjunto de elementos entre los cuales existe alguna preferencia. En este caso, $\mu_V(u)$ representa el grado de preferencia entre dichos elementos. Esto es común en problemas de decisión, donde podemos representar un conjunto de criterios y restricciones mediante conjuntos borrosos, que condicionan el valor que han de tomar determinadas variables de decisión. Esta semántica es propia de tareas de optimización, diseño o planificación.
- *Incertidumbre:* Esta semántica permite a Zadeh introducir una teoría de la posibilidad basada en conjuntos borrosos [Zadeh, 1978]. Dada una variable x que

toma valores en U , el predicado “ x toma un valor pequeño” se modela mediante un conjunto borroso $V \subset U$, con función de pertenencia $\mu_{V=\text{pequeño}}$. La teoría de la posibilidad nos dice que si la única información disponible es que x es pequeño, y su valor preciso es desconocido, entonces $\mu_{V=\text{pequeño}}$ se puede utilizar como una medida de la posibilidad de que el valor de x sea $u \in U$, lo que se representa por $\pi_x(u) = \mu_{V=\text{pequeño}}(u)$, $\forall u \in U$.

Cuando una función de pertenencia modela una distribución de posibilidad los elementos que pertenecen al soporte son candidatos mutuamente excluyentes al valor de x , lo que contrasta con el carácter usual de reunión de la noción de conjunto. Esta semántica está ligada a la presencia de incertidumbre en aquellas tareas de razonamiento que llevan a cabo los sistemas basados en conocimiento.

Proponemos un ejemplo que debemos a Dubois y Prade [Dubois y Prade, 2000], y que servirá para clarificar los usos en los que podemos encontrarnos estas tres semánticas. Supongamos que nos interesa clasificar los coches de un aparcamiento en las categorías “coche grande”, “coche mediano” y “coche pequeño”. Para calcular el grado de pertenencia de un coche particular a la categoría “coche pequeño” podemos utilizar un prototipo de coche pequeño. Cuanto más se acerca el tamaño de un coche al del prototipo, mayor es su grado de pertenencia a la categoría de coche pequeño.

Supongamos que deseamos comprar un coche, y que consideramos un conjunto de criterios que debe satisfacer, uno de los cuales es que su tamaño ha de ser pequeño. En este caso, el grado de pertenencia de un coche a la categoría “coche pequeño” ordena el grado de satisfacción o preferencia de ese coche para un agente decisor, en función de uno de los criterios que maneja. La decisión final ha de considerar alguna fórmula de compromiso entre la satisfacción de los distintos criterios.

Supongamos ahora que nos dicen que han visto un coche pequeño salir del aparcamiento a gran velocidad. Desconocen qué coche es y todo lo que saben de él es que su tamaño es pequeño. En este caso, el grado de pertenencia de un coche a la categoría “coche pequeño” representa el grado de posibilidad de que, a nuestro juicio, un coche así sea el mismo que han visto salir a gran velocidad. Aun cuando este grado de posibilidad sea alto, no tenemos ninguna certeza de qué coche es, sobre todo si el conjunto de coches que encajan con la descripción de “coche pequeño” es numeroso. La función de pertenencia responde a la incertidumbre que existe sobre el coche que ha salido del aparcamiento.

Así todo, las tres semánticas aquí presentadas no son excluyentes. Podemos encontrarnos con distribuciones de posibilidad utilizadas para definir categorías en problemas de clasificación. O con la necesidad de modelar la incertidumbre acerca de una determinada preferencia. No debemos, por tanto, entender estas semánticas como comportamientos estancos, sino como una forma de caracterizar los distintos usos del lenguaje, y que nos han de obligar a realizar un análisis de las propiedades que ha de adoptar en cada uno de sus usos.

7.4 Teorías de conjuntos borrosos

En último término, nuestra intención es proporcionar desde la teoría de conjuntos una representación del significado de aquellas sentencias que aparecen en el lenguaje natural. En el caso particular de los predicados precisos, es posible realizar la composición de predicados mediante el uso de la negación, la conjunción, y la disyunción, que en su representación conjuntista dan lugar a fórmulas que involucran a las operaciones de complemento, intersección y unión.

Así, dado un conjunto $A \subset U$, decimos que “ u es no A ”, si $u \in \overline{A}$, que se define como:

$$\overline{A} = \{u \in U : u \notin A\}$$

Dados dos conjuntos precisos $A \subset U$ y $B \subset U$, decimos que “ u es A y B ” si $u \in A \cap B$, que se define como:

$$A \cap B = \{u \in U : u \in A \text{ y } u \in B\}$$

Dados dos conjuntos $A \subset U$ y $B \subset U$, decimos que “ u es A o B ” si $u \in A \cup B$, que se define como:

$$A \cup B = \{u \in U : u \in A \text{ o } u \in B\}$$

Es un resultado bien conocido en matemáticas que el conjunto $\mathcal{P}(U)$ de partes de U , esto es, el conjunto formado por todos aquellos subconjuntos precisos de un conjunto de referencia U , forma una estructura $(\mathcal{P}(U), \cup, \cap, -)$ denominada *álgebra de Boole*, dotada de un conjunto de propiedades (véase la Tabla 7.1) de entre las que destacamos la *ley del tercero excluido* $A \cup (\overline{A}) = U$, que nos dice que un predicado sólo puede ser cierto o falso, y el *principio de no contradicción* $A \cap (\overline{A}) = \emptyset$, que nos dice que un predicado no puede ser al mismo tiempo cierto y falso. Estas propiedades resultan fundamentales para asegurar la consistencia de nuestros razonamientos.

Idempotencia:	$A \cup A = A$	$A \cap A = A$
Commutativa:	$A \cup B = B \cup A$	$A \cap B = B \cap A$
Asociativa:	$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cap (B \cap C) = (A \cap B) \cap C$
Absorción:	$A \cup (A \cap B) = A$	$A \cap (A \cup B) = A$
Distributiva:	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identidad:	$A \cup \emptyset = A$	$A \cap U = A$
Complemento:	$A \cup \overline{A} = U$	$A \cap \overline{A} = \emptyset$
De Morgan:	$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$	$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$
Involución:		$\overline{\overline{A}} = A$

Tabla 7.1: Propiedades de las álgebras de Boole.

También podemos representar la composición de predicados precisos utilizando funciones de pertenencia, haciendo uso de las siguientes definiciones:

$$\mu_{\overline{A}}(u) = 1 - \mu_A(u)$$

$$\mu_{A \cap B}(u) = \min\{\mu_A(u), \mu_B(u)\}$$

$$\mu_{A \cup B}(u) = \max\{\mu_A(u), \mu_B(u)\}$$

Estas definiciones nos permiten asegurar que resulta equivalente operar conjuntos mediante la intersección, la unión y el complemento, a hacerlo con funciones de pertenencia mediante el mínimo, el máximo y el complemento a uno. Dicho de otro modo, si llamamos $\{0, 1\}^U$ al conjunto de funciones $\mu : U \rightarrow \{0, 1\}$, decimos que las álgebras de Boole $(\mathcal{P}(U), \cup, \cap, -)$ y $(\{0, 1\}^U, \max, \min, 1 - x)$ son isomorfas.

Parece natural buscar ahora una representación matemática para la composición de predicados borrosos mediante la negación, la conjunción y la disyunción. Además, debemos exigir que dicha representación funcione como una extensión que ha de permitirnos recuperar las expresiones anteriores en el caso de que los predicados involucrados sean precisos. Una forma de representar esta composición es mediante el uso de funciones: $n : [0, 1] \rightarrow [0, 1]$, $T : [0, 1]^2 \rightarrow [0, 1]$ y $S : [0, 1]^2 \rightarrow [0, 1]$ de manera que:

$$\begin{aligned}\mu_{\overline{A}}(u) &= n(\mu_A(u)) \\ \mu_{A \cap_T B}(u) &= T(\mu_A(u), \mu_B(u)) \\ \mu_{A \cup_S B}(u) &= S(\mu_A(u), \mu_B(u))\end{aligned}$$

donde suponemos que tanto A como B son subconjuntos borrosos de U . A estas funciones les hemos de pedir que respondan a nuestras intuiciones cuando manipulamos predicados vagos, y que proporcionen resultados que nos parezcan razonables.

Esto nos lleva a exigir que la función n satisfaga que $n(0) = 1$ y $n(1) = 0$, y además:

$$\forall x, y \in [0, 1], x \leq y \Rightarrow n(x) \geq n(y)$$

Una función n con estas propiedades se denomina *negación*. Existen, desde luego, muchas funciones que pueden jugar el papel de negaciones. Por ejemplo, $n(x) = \sqrt{1 - x^2}$ o $n(x) = (1 - t)/(1 + \lambda t)$, siendo $\lambda \geq -1$. Sin embargo, es común tomar $n(x) = 1 - x$. Esta negación es continua en x , y verifica además que $n(n(x)) = x$, $\forall x \in [0, 1]$, lo que se corresponde con nuestra intuición de que $\mu_{\overline{\overline{A}}} = \mu_A$. Estas características hacen que reciba el nombre particular de *negación fuerte*.

Para las funciones T y S podríamos pedir las siguientes propiedades:

1. *Ley de identidad*: $\forall x \in [0, 1]$

$$T(x, 1) = x \quad (A \cap U = A)$$

$$S(x, 0) = x \quad (A \cup \emptyset = A)$$

2. *Ley conmutativa*: $\forall x, y \in [0, 1]$

$$T(x, y) = T(y, x) \quad (A \cap B = B \cap A)$$

$$S(x, y) = S(y, x) \quad (A \cup B = B \cup A)$$

3. *Ley asociativa*: $\forall x, y, z \in [0, 1]$

$$T(x, T(y, z)) = T(T(x, y), z) \quad (A \cap (B \cap C) = (A \cap B) \cap C)$$

$$S(x, S(y, z)) = S(S(x, y), z) \quad (A \cup (B \cup C) = (A \cup B) \cup C)$$

4. *Ley de monotonía:* $\forall x, y, u, v \in [0, 1], x \leq u, y \leq v$

$$T(x, y) \leq T(u, v)$$

$$S(x, y) \leq S(u, v)$$

En matemáticas, aquellas funciones T que satisfacen las anteriores propiedades reciben el nombre de *normas triangulares* o t-normas; y aquellas funciones S que también las satisfacen reciben el nombre de *conormas triangulares* o t-conormas.

A partir de las leyes de identidad y monotonía podemos deducir para cualquier $x \in [0, 1]$:

$$T(0, x) = 0 \quad y \quad S(1, x) = 1$$

Lo que nos parece muy razonable. Pero además, podemos deducir de las mismas leyes una condición que han de satisfacer cualesquiera operadores T y S que se nos puedan ocurrir para realizar la conjunción y disyunción de predicados:

$$T(x, y) \leq \min(x, y) \quad y \quad S(x, y) \geq \max(x, y)$$

Luego el mínimo es la mayor de las t-normas y el máximo es la menor de las t-conormas.

A partir de una t-norma T y una negación fuerte n , podemos definir una operación S mediante la siguiente expresión:

$$S(x, y) = n(T(n(x), n(y))), \quad \forall x, y \in [0, 1]$$

Así definida, S es una t-conorma. Además, se cumple que $T(x, y) = n(S(n(x), n(y)))$, $\forall x, y \in [0, 1]$. Obtenemos así una expresión para las leyes de De Morgan en el caso de conjuntos borrosos. S se conoce como la t-conorma n-dual de T , y T como la t-norma n-dual de S . Cuando $n(x) = 1 - x$ entonces hablamos simplemente de la t-conorma dual y la t-norma dual.

En la bibliografía podemos encontrar múltiples propuestas de pares de t-normas y t-conormas duales que satisfacen las propiedades anteriores, y de entre las que destacamos las siguientes:

- *El máximo y el mínimo:* son las propuestas por Zadeh en su trabajo fundacional de 1965:

$$T_Z(x, y) = \min(x, y) \quad S_Z(x, y) = \max(x, y)$$

- *El producto y la suma probabilística:* propuestas por Goguen en 1969 [Goguen, 1969]:

$$T_G(x, y) = xy \quad S_G(x, y) = x + y - xy$$

- *Las t-norma y t-conorma de Lukasiewicz:* introducidas por Lukasiewicz en 1931 [Lukasiewicz, 1970]:

$$T_L(x, y) = \max(x + y - 1, 0) \quad S_L(x, y) = \min(x + y, 1)$$

Para cada problema es posible escoger una combinación cualquiera de una t-norma y una t-conorma (nueve posibilidades si nos atenemos a las que aquí mostramos, y prescindiendo de la dualidad). Ahora bien, cada una de ellas determina unas propiedades particulares en el cálculo con conjuntos borrosos, lo que condiciona el significado y uso de los predicados que obtenemos como resultado.

Por ejemplo, las leyes del tercero excluido y de no contradicción no se verifican ni con el par máximo y mínimo, ni con el producto y suma probabilística. Veámoslo:

$$\mu_{A \cap \overline{A}}(u) = \min\{\mu_A(u), 1 - \mu_A(u)\} \quad (\text{t-norma mínimo})$$

$$\mu_{A \cap \overline{A}}(u) = \mu_A(u) \cdot (1 - \mu_A(u)) \quad (\text{t-norma producto})$$

Sólo son constantes e iguales a cero si $\mu_A(u)$ es igual a cero o igual a uno para todo $u \in U$. Del mismo modo:

$$\mu_{A \cup \overline{A}}(u) = \max\{\mu_A(u), 1 - \mu_A(u)\} \quad (\text{t-conorma máximo})$$

$$\mu_{A \cup \overline{A}}(u) = \mu_A(u) + 1 - \mu_A(u) - \mu_A(u)(1 - \mu_A(u)) \quad (\text{t-conorma suma probabilística})$$

Sólo son constantes e iguales a uno si $\mu_A(u)$ es igual a cero o a igual a uno para todo $u \in U$. Sin embargo, si utilizamos la t-norma y la t-conorma de Lukasiewicz sí se satisfacen estas dos leyes:

$$\mu_{A \cap \overline{A}}(u) = \max\{0, \mu_A(u) + 1 - \mu_A(u) - 1\} = 0$$

$$\mu_{A \cup \overline{A}}(u) = \min\{1, \mu_A(u) + 1 - \mu_A(u)\} = 1$$

Por tanto, si necesitamos que se satisfaga la ley del tercero excluido debemos utilizar como t-conorma la de Lukasiewicz, junto con otra t-norma cualquiera. Si necesitamos que se satisfaga el principio de no contradicción debemos utilizar la t-norma de Lukasiewicz, junto con otra t-conorma cualquiera. Si se desean ambas leyes se deben utilizar tanto la t-norma como la t-conorma de Lukasiewicz.

Las cosas se complican si consideramos que se han de satisfacer otras propiedades. Tomemos por ejemplo, las propiedades distributivas, que resultan muy importantes en el cálculo con conjuntos precisos:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Si estimamos necesarias ambas propiedades distributivas en el caso de conjuntos borrosos, la única posibilidad es utilizar simultáneamente la t-norma mínimo y la t-conorma máximo, ya que cualquier otra pareja no la satisface.

Por otra parte, si consideramos la definición de conjunto borroso haciendo uso de sus α -cortes, $A_\alpha = \{u \in U | \mu_A(u) \geq \alpha\}$, entonces para dos conjuntos borrosos $A, B \subset U$, las igualdades $(A \cup B)_\alpha = A_\alpha \cup B_\alpha$ y $(A \cap B)_\alpha = A_\alpha \cap B_\alpha$ sólo son ciertas si utilizamos la t-norma mínimo, la t-conorma máximo y la negación $1 - x$. Sin embargo, $(\overline{A})_\alpha \neq (\overline{A_\alpha})$ cuando $\alpha \neq 1$, ya que, en general, hay elementos que no pertenecen ni a A_α ni a $(\overline{A})_\alpha$, esto es, $A_\alpha \cup (\overline{A})_\alpha \neq U$.

Una vez que caracterizamos las distintas operaciones conjuntistas que podemos realizar sobre conjuntos borrosos, podemos definir una teoría de conjuntos borrosos:

Definición 7.1. Podemos definir una teoría de conjuntos borrosos como la tupla $([0, 1]^U, T, S, n)$, donde $[0, 1]^U = \{\mu \mid \mu : U \rightarrow [0, 1]\}$ es el conjunto de funciones de pertenencia entre U y $[0, 1]$, T es una t-norma, S es una t-conorma, y n es una negación fuerte.

La estructura algebraica originada no es un álgebra de Boole, salvo en el subconjunto $\{0, 1\}^U$.

Vemos, por tanto, que no existe una única teoría de conjuntos borrosos, sino múltiples teorías, resultado de una elección de aquellas propiedades deseables en los distintos usos del lenguaje. Puesto que los conjuntos borrosos nos permiten formalizar el significado de los predicados vagos que utilizamos en el lenguaje, esto nos puede sugerir que cada hablante incorpora alguna teoría particular de la significación que proviene del aprendizaje, y que obedece a aquellas propiedades o leyes que le han de permitir razonar mediante el lenguaje de un modo consistente.

Por otra parte, el significado de un predicado tiene no sólo un carácter subjetivo, sino también local, en el sentido de que su validez está restringida a un uso particular del lenguaje. Así, la definición que hacemos del predicado “alto” mediante un determinado conjunto borroso puede variar según el contexto en el que hablamos de la altura: una clase de educación infantil, un equipo de baloncesto, etc. Por tanto, asignaremos una función de pertenencia a un uso particular de un determinado predicado. Lo mismo se puede decir de la conjunción, la disyunción y la negación, de tal manera que podemos ir más allá y decir que en los distintos usos del lenguaje son múltiples las teorías de conjuntos borrosos que nos han de permitir analizar el significado del discurso.

7.5 Variable lingüística

La noción de variable lingüística fue introducida por Zadeh en 1975 [Zadeh, 1975]. Una variable lingüística V toma valores en un conjunto $\mathcal{L} = \{L_1, L_2, \dots, L_n\}$ de términos lingüísticos que realizan una descripción cualitativa de un conjunto de referencia U . Por ejemplo, si U es el conjunto de temperaturas, podríamos pensar en una descripción $\mathcal{L}_T = \{\text{frío}, \text{templado}, \text{caliente}\}$. Cada uno de estos términos $L_i \in \mathcal{L}$ puede representarse mediante un conjunto borroso de U , donde normalmente U se corresponde con una escala de valores numéricos.

Podemos añadir a estos términos lingüísticos el uso de *modificadores lingüísticos*, como son ‘muy’, ‘bastante’, ‘más o menos’, etc., que nos permiten enriquecer el lenguaje. Sea μ_{L_i} la función de pertenencia asociada al término L_i . Podemos modelar la acción de un modificador lingüístico M_i sobre la etiqueta L_i mediante dos funciones: una función de modificación $m_{M_i} : [0, 1] \rightarrow [0, 1]$, y una función de desplazamiento $d_{M_i} : U \rightarrow U$, de modo que:

$$\mu_{M_i L_i} = m_{M_i} \circ \mu_{L_i} \circ d_{M_i}$$

donde \circ representa la composición de funciones [Lakoff, 1973]. Un modificador lingüístico como ‘muy’ denota en ciertos usos un incremento en la precisión, como puede

ser en ‘‘muy cierto’’ o ‘‘muy rojo’’, lo que podemos modelar haciendo d_{muy} igual a la función identidad I , y exigiendo que $m_{muy} \circ \mu_{L_i} \circ I \leq \mu_{L_i}$. Una opción es definir $m_{muy} = x^2$, $x \in [0, 1]$, de modo que $\mu_{M_i L_i} = (\mu_{L_i})^2$. En otros usos, ‘muy’ parece denotar una traslación en el conjunto de referencia U , como puede ser en ‘‘muy caliente’’ o ‘‘muy pequeño’’, lo que podemos modelar haciendo m_{muy} igual a la función identidad I , y $d_{muy}(u) = d + u$ (véase la Figura 7.3). El modificador lingüístico ‘más o menos’ parece denotar una pérdida de precisión, lo que podemos modelar haciendo $m_{más\ o\ menos} \circ \mu_{L_i} \geq \mu_{L_i}$. Una opción es definir $m_{más\ o\ menos} = \sqrt{x}$, $x \in [0, 1]$, de modo que $\mu_{M_i L_i} = \sqrt{\mu_{L_i}}$. También la negación se puede modelar como un modificador



Figura 7.3: Un ejemplo de a) el efecto de la función de modificación y b) de la función de desplazamiento en el uso de modificadores lingüísticos.

lingüístico, haciendo $m_{no} \circ \mu_{L_i} = 1 - \mu_{L_i}$. En el caso particular de que el conjunto de referencia U se corresponda con una escala numérica restringida a un intervalo $[a, b]$, podemos definir el antónimo de cualquiera de los términos lingüísticos de \mathcal{L} haciendo uso de la simetría con respecto al punto medio de U , esto es:

$$\mu_{ant(L_i)}(u) = \mu_{L_i}(b + a - u)$$

En particular, si $[a, b] = [0, 1]$, entonces $\mu_{ant(L_i)}(u) = \mu_{L_i}(1 - u)$.

Una vez que representamos cada uno de los términos lingüísticos de \mathcal{L} mediante conjuntos borrosos, decimos que éstos forman una *partición borrosa* de U si se satisfacen dos condiciones (véase la Figura 7.4):

1. *Cobertura*: $\forall u \in U, \max_{i=1,\dots,n} \mu_{L_i}(u) > 0$
2. *Exclusión mutua*: $\forall u \in U, \forall i \neq j, \min\{\mu_{L_i}(u), \mu_{L_j}(u)\} < 1$

Aunque en muchas ocasiones se exige una condición más restrictiva, debida a Ruspini [Ruspini, 1969], y que nos dice que $\forall u \in U$:

$$\sum_{i=1}^n \mu_{L_i}(u) = 1$$

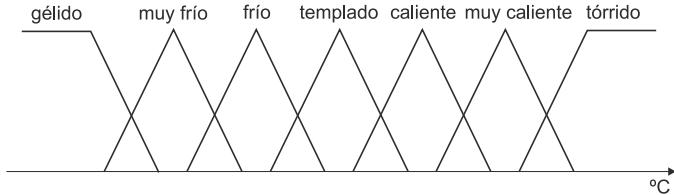


Figura 7.4: Un ejemplo de partición borrosa en el conjunto de las temperaturas.

7.6 Principio de extensión

Muchas de las ideas con las que jugamos en este capítulo son una extensión al lenguaje borroso de conceptos que en el lenguaje de lo preciso resultan de manifiesta utilidad. En este sentido, una de las herramientas más importantes en el ámbito de los conjuntos borrosos es el *principio de extensión* de Zadeh. Sea una función $\phi : U_1 \times \dots \times U_n \rightarrow W$, que tiene como argumentos n valores precisos $(u_1, \dots, u_n) \in U_1 \times \dots \times U_n$, y tal que $w = \phi(u_1, \dots, u_n)$. Este principio nos permite extender la función ϕ a un conjunto de argumentos borrosos $A_1 \subset U_1, \dots, A_n \subset U_n$, e inferir un nuevo conjunto borroso $B \subset W$ como $B = \Phi(A_1, \dots, A_n)$, mediante la siguiente expresión:

$$\begin{aligned}\mu_B(w) &= \sup_{w=\phi(u_1, u_2, \dots, u_n)} \min\{\mu_{A_1}(u_1), \mu_{A_2}(u_2), \dots, \mu_{A_n}(u_n)\} \\ &= 0 \quad \text{si } \phi^{-1}(w) = \emptyset.\end{aligned}$$

siendo μ_{A_i} la función de pertenencia de A_i ($i = 1, \dots, n$). De este modo, el principio de extensión nos permite hacer borroso cualquier dominio del razonamiento matemático basado en la teoría de conjuntos. Además, toda extensión borrosa ha de satisfacer siempre un axioma básico: ha de preservar el resultado clásico cuando opera sobre conjuntos precisos.

Un ejemplo de aplicación del principio de extensión nos permite definir un *grado de compatibilidad* entre conjuntos borrosos. Dado un conjunto borroso $A \subset U$, $\mu_A(u)$ es el grado de pertenencia de u a A . Pongamos que A modela el significado del término lingüístico ‘alto’. Podemos interpretar $\mu_A(u)$ como el grado de compatibilidad de u con el significado de ‘alto’. Hasta aquí nada nuevo. Lo que nos proporciona el principio de extensión es la capacidad de evaluar la compatibilidad del conjunto borroso A con otro conjunto borroso B tomado como referencia. Si llamamos $C(A, B)$ a esta compatibilidad, podemos pensar en $C(A, B)$ como en un conjunto borroso cuya función de pertenencia $\mu_{C(A,B)} : [0, 1] \rightarrow [0, 1]$ se interpreta como $\mu_{C(A,B)} = \mu_A(B)$, y que podemos obtener $\forall v \in [0, 1]$ mediante la expresión (véase la Figura 7.5):

$$\mu_{C(A,B)}(v) = \sup_{u: v=\mu_A(u)} \mu_B(u)$$

Si $A = B$, se cumple que $\mu_{C(A,B)}(v) = v$, donde $v \in [0, 1]$, indicando que dos conjuntos borrosos iguales son compatibles. Si $A = \overline{B}$, entonces $\mu_{C(A,B)}(v) = 1 - v$, lo que indica

que dos conjuntos complementarios son incompatibles. Además, si $B \subseteq A$, entonces $\mu_{C(A,B)}(v) \leq v$, lo que nos permite pensar en la compatibilidad entre conjuntos borrosos como en un índice de inclusión borrosa.

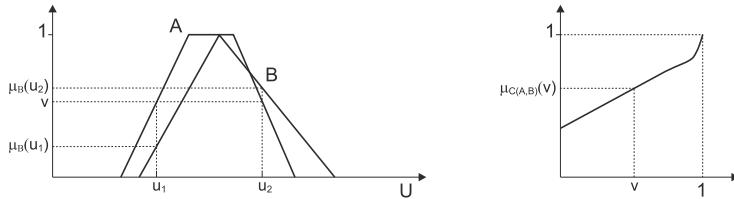


Figura 7.5: Un ejemplo de cálculo de la compatibilidad entre conjuntos borrosos.

7.6.1 Aritmética borrosa

Podemos ilustrar el principio de extensión en el dominio de la aritmética. Definamos un *número borroso* como un conjunto borroso convexo y normalizado en el conjunto de los números reales \mathbb{R} . Decimos que un número borroso A es *positivo* si $\mu_A(x) = 0, \forall x < 0$. Del mismo modo, un número borroso A es *negativo* si $\mu_A(x) = 0, \forall x > 0$.

Sean $A, B \subset \mathbb{R}$ dos de estos números borrosos. El principio de extensión nos dice que podemos definir su *suma borrosa*, $A \oplus B$, mediante la siguiente expresión:

$$\mu_{A \oplus B}(z) = \sup_{z=x+y} \min\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

La suma borrosa de dos números borrosos también es un número borroso. También podemos definirla haciendo uso de la representación en α -cortes, de modo que $(A \oplus B)_\alpha = A_\alpha + B_\alpha, \forall \alpha \in [0, 1]$. Siendo A y B números borrosos, sus α -cortes son intervalos cerrados ($A_\alpha = [a_\alpha^1, a_\alpha^2]$ y $B_\alpha = [b_\alpha^1, b_\alpha^2]$), lo que permite descomponer la suma borrosa en una suma de intervalos $A_\alpha + B_\alpha = [a_\alpha^1 + b_\alpha^1, a_\alpha^2 + b_\alpha^2]$. El elemento neutro de la suma es el número preciso 0, y satisface que $A \oplus 0 = 0 \oplus A = A$. Se puede demostrar de un modo trivial que la suma borrosa satisface las propiedades conmutativa y asociativa.

De un modo análogo podemos extender el resto de las operaciones aritméticas. Así, la *resta borrosa* de dos números borrosos A y B es un número borroso $A \ominus B$ que obedece a la siguiente expresión:

$$\mu_{A \ominus B}(z) = \sup_{z=x+y} \min\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

También podemos representar esta operación mediante aritmética de intervalos, adoptando la forma $(A \ominus B)_\alpha = [a_\alpha^1 - b_\alpha^2, a_\alpha^2 - b_\alpha^1], \forall \alpha \in [0, 1]$. Si definimos el *opuesto* de un número borroso B como el número borroso B^- , tal que $\mu_{B^-}(x) = \mu_B(-x)$, $\forall x \in \mathbb{R}$, es trivial comprobar que $A \ominus B = A \oplus B^-$. Debemos hacer notar que el

propio opuesto puede definirse como extensión borrosa del operador unitario que nos permite obtener el opuesto de un número preciso. Por otra parte, es necesario tener cuidado con las operaciones de la aritmética borrosa, ya que no satisfacen las mismas propiedades que sus correspondientes en la aritmética precisa. Así, por ejemplo, dado un conjunto borroso A , se tiene que $A \ominus A \neq 0$, y por la misma razón $(A \ominus B) \oplus B \neq A$. La resta borrosa no satisface ni la propiedad conmutativa ni la asociativa.

La *multiplicación borrosa* de dos números borrosos A y B es un número borroso $A \otimes B$ que viene dado por la siguiente expresión:

$$\mu_{A \otimes B}(z) = \sup_{z=x \times y} \min\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

La representación de esta operación mediante aritmética de intervalos adopta la forma $(A \otimes B)_\alpha = [\min(a_\alpha^1 \times b_\alpha^1, a_\alpha^1 \times b_\alpha^2, a_\alpha^2 \times b_\alpha^1, a_\alpha^2 \times b_\alpha^2), \max(a_\alpha^1 \times b_\alpha^1, a_\alpha^1 \times b_\alpha^2, a_\alpha^2 \times b_\alpha^1, a_\alpha^2 \times b_\alpha^2)]$, $\forall \alpha \in [0, 1]$. El elemento neutro de la multiplicación borrosa es el número preciso 1, y satisface que $1 \otimes A = A \otimes 1 = A$. Se puede demostrar de un modo trivial que la multiplicación borrosa satisface las propiedades conmutativa y asociativa. Sin embargo, no satisface en general la propiedad distributiva con respecto a la suma, sino una versión más débil: $A \otimes (B \oplus C) \subseteq (A \otimes B) \oplus (A \otimes C)$. La propiedad distributiva se cumple si A es positivo o negativo, y B y C son al mismo tiempo positivos o negativos.

La *división borrosa* de dos números borrosos A y B es un número borroso $A \oslash B$ que viene dado por la siguiente expresión:

$$\mu_{A \oslash B}(z) = \sup_{z=x/y} \min\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}^+$$

Podemos representar esta operación en \mathbb{R}^+ mediante aritmética de intervalos, adoptando la forma $(A \oslash B)_\alpha = [a_\alpha^1/b_\alpha^2, a_\alpha^2/b_\alpha^1]$, $\forall \alpha \in [0, 1]$, donde debe satisfacerse que $b_\alpha^2 > 0$. Si definimos el *inverso* de un número borroso B como el número borroso B^{-1} , tal que $\mu_{B^{-1}}(x) = \mu_B(1/x)$, $\forall x \in \mathbb{R} - \{0\}$, es trivial comprobar que $A \oslash B = A \otimes B^{-1}$. Sin embargo, $A \oslash A \neq 1$, y por la misma razón $(A \oslash B) \otimes B \neq A$. La división borrosa no satisface las propiedades conmutativa y asociativa. Hasta ahora hemos restringido nuestra discusión sobre la división a \mathbb{R}^+ . $A \oslash B$ es un número borroso si tanto A como B son positivos o negativos. En cambio, si un número borroso B no es ni positivo ni negativo, entonces B^{-1} no es un número borroso. En el caso de que tanto A como B no estén restringidos a \mathbb{R}^+ la división borrosa ha de realizarse como composición, donde la parte positiva de $A \oslash B$ resulta de la unión de la división de las partes positivas y la división de las partes negativas de A y B , y la parte negativa de $A \oslash B$ resulta de la unión de la división de la parte positiva de A y la parte negativa de B y la división de la parte negativa de A y la parte positiva de B .

La Figura 7.6 muestra un ejemplo de la aplicación de las operaciones de la aritmética borrosa a un par de números borrosos definidos en \mathbb{R}^+ , un número borroso A que podríamos definir como “aproximadamente 2”, y un número borroso B que podríamos definir como “aproximadamente 3”.

Vemos cómo los resultados confirman nuestras intuiciones, de modo que $A \oplus B$ podríamos decir que es “aproximadamente 5”, $B \ominus A$ es “aproximadamente 1”, $A \otimes B$ es “aproximadamente 6”, y $B \oslash A$ es “aproximadamente 1’5”. También observamos

en la misma Figura 7.6 cómo el resultado de estas operaciones es un número borroso cuya imprecisión aumenta respecto a los números borrosos que operamos. El principio

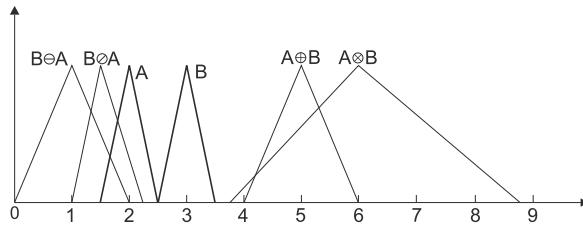


Figura 7.6: Un ejemplo de aplicación de operaciones de aritmética borrosa.

de extensión también es aplicable a otras funciones matemáticas, como es el caso de $f(x) = e^x$, cuya extensión borrosa se puede definir como:

$$\mu_{e^A}(x) = \begin{cases} \mu_A(\ln(x)), & \text{si } x > 0, \\ 0, & \text{en otro caso.} \end{cases}$$

e^A es un número borroso positivo. Además, se cumple que $e^{-A} = (e^A)^{-1}$, y $e^A \otimes e^B = e^{A \oplus B}$.

7.7 Relaciones borrosas

Las relaciones borrosas permiten modelar sentencias del tipo de “es mucho más alto que” o “es muy parecido a”. Definimos una *relación borrosa* como un subconjunto borroso del producto cartesiano entre conjuntos. Así, por ejemplo, dados dos conjuntos U y V , definimos la relación R mediante una función de pertenencia $\mu_R : U \times V \rightarrow [0, 1]$, de manera que para todo par $(u, v) \in U \times V$, $\mu_R(u, v)$ determina el grado de relación que existe entre u y v .

Supongamos que $U = \{u_1, u_2, u_3, u_4\}$ y $V = \{v_1, v_2, v_3, v_4\}$. Podemos definir entre ellos la relación R dada por:

$$\begin{pmatrix} 1 & 0,2 & 0,2 & 0,8 \\ 0,2 & 1 & 0,6 & 0,2 \\ 0,2 & 0,6 & 1 & 0,2 \\ 0,8 & 0,2 & 0,2 & 1 \end{pmatrix}$$

Esta relación nos dice que, por ejemplo, $\mu_R(u_2, v_4) = 0,2$, y $\mu_R(u_4, v_1) = 0,8$. Esta definición conjuntista de las relaciones borrosas nos permite realizar con las relaciones las mismas operaciones conjuntistas que hemos definido para los conjuntos borrosos. Podemos, por tanto, hablar de una unión, una intersección o del complemento de relaciones borrosas. Sin embargo, si consideramos la definición de una relación borrosa haciendo uso de sus α -cortes, $R_\alpha = \{(u, v) \in U \times V | \mu_R(u, v) \geq \alpha\}$, entonces para dos

relaciones borrosas R y Q las igualdades $(R \cup Q)_\alpha = R_\alpha \cup Q_\alpha$ y $(R \cap Q)_\alpha = R_\alpha \cap Q_\alpha$ sólo son ciertas si utilizamos la t-norma mínimo, la t-conorma máximo y la negación $1 - x$. Este resultado es una generalización del mostrado en la sección 7.4 para el caso de conjuntos borrosos.

7.7.1 Relaciones de similitud

Un tipo de relaciones borrosas especialmente útil es el de las *relaciones de similitud* o de equivalencia, definidas mediante una función de pertenencia $\mu_S : U \times U \rightarrow [0, 1]$, donde U se corresponde habitualmente con una escala numérica. Todas las relaciones de similitud deben satisfacer tres propiedades:

S1. *Reflexiva*: $\forall u \in U, \mu_S(u, u) = 1$.

S2. *Simétrica*: $\forall u, v \in U, \mu_S(u, v) = \mu_S(v, u)$.

S3. *t-Transitiva*: $\forall u, v, w \in U, T(\mu_S(u, v), \mu_S(v, w)) \leq \mu_S(u, w)$.

donde T denota una t-norma cualquiera, haciendo depender la propiedad de transitividad de la t-norma escogida. Es muy fácil comprobar que la matriz ejemplo que proponemos arriba se corresponde con una relación de similitud.

La noción de similitud es complementaria a la de diferencia, que formalmente se modela mediante alguna distancia $d(u, v)$ entre los elementos u y v . Esto nos da una pista sobre el posible diseño de una relación de similitud adecuada a los propósitos de un determinado problema. Podemos modelar una similitud mediante dos funciones: una distancia $d : U \times U \rightarrow [0, +\infty)$, y una función de proximidad $\mu_p : [0, +\infty) \rightarrow [0, 1]$, función no creciente a la que se exige que $\mu_p(0) = 1$, de modo que:

$$\mu_S = \mu_p \circ d$$

Dubois y Prade proponen un ejemplo de relación de similitud en la que $d(u, v) = |u - v|$ (véase la Figura 7.7):

$$\begin{aligned} \mu_S(u, v) &= \max\left(0, \min\left(1, \frac{\delta + \rho - |u - v|}{\rho}\right)\right) \\ &= \begin{cases} 1, & \text{si } |u - v| \leq \delta, \\ 0, & \text{si } |u - v| \geq \delta + \rho, \\ (\delta + \rho - |u - v|)/\rho, & \text{en otro caso.} \end{cases} \end{aligned}$$

donde $\rho > 0$ y $\delta \geq 0$ [Dubois y Prade, 1989]. Estos dos parámetros permiten ajustar el significado de la similitud en relación con la proximidad de los elementos u y v . Así, distintos valores de ρ y δ pueden determinar el significado de términos como “muy similares” o “más o menos similares”, lo que también es posible hacer mediante el uso de los correspondientes modificadores lingüísticos sobre la función de proximidad. Por otra parte, y como era de esperar, la función de pertenencia obtenida como $\mu_D(u, v) = 1 - \mu_S(u, v)$ modela una relación de diferencia o distancia entre los elementos u y v (véase ejercicio resuelto 3).

Una relación de similitud resulta una herramienta muy útil en la ingeniería. Pensemos que tareas como el reconocimiento de patrones, la clasificación, el agrupamiento o la recuperación de información, entre otras, utilizan alguna medida de similitud con respecto a un conjunto de prototipos, y su modelado borroso permite realizar una ordenación de los objetos del problema según un criterio de preferencia que viene determinado por la forma de la relación borrosa de similitud.

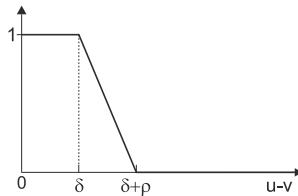


Figura 7.7: Un ejemplo de relación borrosa de similitud.

7.7.2 Relaciones de comparación

De un modo análogo podemos modelar *relaciones de comparación*, como pueden ser “mucho mayor que” o “menor o igual que”, mediante una expresión del tipo de:

$$\mu_C = 1 - \mu_p \circ \max(0, u - v)$$

donde la operación de máximo nos permite asegurar que estamos en el dominio de μ_p . Así, por ejemplo, podríamos modelar la relación “mayor que” mediante la siguiente expresión:

$$\mu_C(u, v) = \min\left(1, \max\left(0, \frac{u - v - \delta}{\rho}\right)\right) = \begin{cases} 1, & \text{si } u > v + \delta + \rho, \\ 0, & \text{si } u < v + \delta, \\ (u - v - \delta)/\rho, & \text{en otro caso.} \end{cases}$$

donde $\rho > 0$ y $\delta \geq 0$. Si hacemos $\delta = 0$ y $\rho \rightarrow 0$ entonces recuperamos la desigualdad ordinaria, con lo que $\mu_C(u, v) = 1$ si $u > v$ y $\mu_C(u, v) = 0$ si $u < v$. De un modo análogo a como razonábamos antes, los dos parámetros $\rho > 0$ y δ permiten ajustar el significado de la comparación en relación con la proximidad de los elementos u y v . Así, distintos valores de ρ y δ pueden determinar el significado de términos como “mucho mayor” o “ligeramente mayor”, lo que también es posible hacer mediante el uso de los correspondientes modificadores lingüísticos sobre la función de proximidad. Además, si C modela la relación “mayor que”, entonces C^I (resultado de intercambiar los argumentos de la relación) modela la relación “menor que”, $C \cup C^I$ es equivalente a la relación de diferencia D , y $\overline{C \cup C^I}$ es equivalente a la relación de similitud S .

7.7.3 Proyección. Extensión cilíndrica

Dos conceptos matemáticos que nos serán de utilidad para la manipulación de información borrosa son los de proyección y extensión cilíndrica. Sea R una relación borrosa en el producto cartesiano $U_1 \times \dots \times U_n$ que asocia, a cada vector $u = (u_1, \dots, u_n) \in U_1 \times \dots \times U_n$ un grado de pertenencia que podemos interpretar como un grado de satisfacción de la relación R . Definimos la *proyección* de R en $U_{i_1} \times \dots \times U_{i_k}$, donde (i_1, \dots, i_k) es una subsecuencia de $(1, 2, \dots, n)$, como una nueva relación borrosa que definimos como:

$$\mu_{Pr(R; U_{i_1} \times \dots \times U_{i_k})}(u_{i_1}, \dots, u_{i_k}) = \sup_{u: U_{i_1} = u_{i_1}, \dots, U_{i_k} = u_{i_k}} \mu_R(u)$$

En el caso particular de que realicemos la proyección de R en U_i , obtenemos el conjunto borroso dado por la siguiente expresión:

$$\mu_{Pr(R; U_i)}(u_i) = \sup_{u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_n} \mu_R(u)$$

Sea R una relación en $U_{i_1} \times \dots \times U_{i_k}$. Definimos la *extensión cilíndrica* de R en $U_1 \times \dots \times U_n$, siendo $(i_1, \dots, i_k) \subseteq (1, \dots, n)$, como una nueva relación borrosa que definimos como:

$$\mu_{Ex(R)}(u) = \mu_R(u_{i_1}, \dots, u_{i_k})$$

Sea la relación borrosa R en $U_1 \times \dots \times U_r$, y la relación borrosa S en $U_s \times \dots \times U_n$, donde $s \leq r + 1$. Definimos la *combinación* de R y S como $Ex(R) \cap Ex(S)$, siendo $Ex(R)$ y $Ex(S)$ extensiones cilíndricas de R y S en $U_1 \times \dots \times U_n$.

Una relación particular entre conjuntos borrosos $A_1 \subset U_1, \dots, A_n \subset U_n$ es su *producto cartesiano*, $A_1 \times \dots \times A_n$, que definimos como un subconjunto borroso de $U_1 \times \dots \times U_n$, cuya función de pertenencia viene dada por:

$$\mu_{A_1 \times \dots \times A_n}(u) = \min(\mu_{A_1}(u_1), \dots, \mu_{A_n}(u_n))$$

Decimos que una relación borrosa R en $U_1 \times \dots \times U_n$ es *separable* si $R(u_1, \dots, u_n) = R(u_1) \times \dots \times R(u_n)$, donde $R(u_i)$ es la proyección de R en U_i , esto es, $R(u_i) = Pr(R; U_i)$. Así, podemos escribir:

$$R = \bigcap_{i=1, n} Ex(Pr(R; U_i))$$

Supongamos que estamos ante un problema que requiere asignar a un conjunto de variables X_1, \dots, X_n , un conjunto de valores $(u_1, \dots, u_n) \in U_1 \times \dots \times U_n$, entre los que existe una relación R . Si R es separable decimos que las variables X_1, \dots, X_n no *interaccionan* entre sí, y podemos asignar de un modo independiente $X_1 = u_1, \dots, X_n = u_n$. En caso contrario, una asignación $X_i = u_i$ depende de las realizadas con anterioridad. La Figura 7.8 ejemplifica el concepto de interacción entre variables.

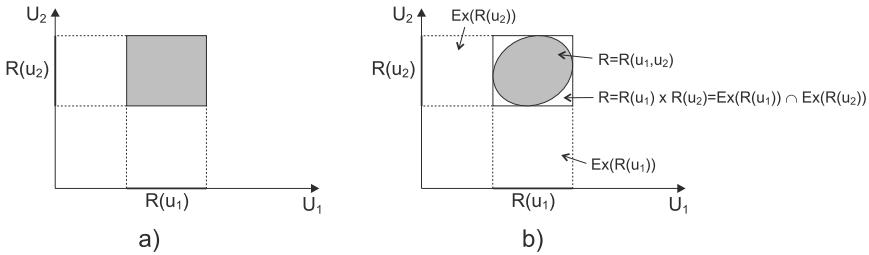


Figura 7.8: En a) se muestra un par de variables que no interaccionan bajo una relación R separable. En b) se muestra un par de variables que interactúan bajo una relación S no separable.

7.7.4 Composición de relaciones

Una operación propia de las relaciones que resulta de especial importancia es la *composición de relaciones*. Dada una relación R entre los conjuntos U y V , y otra relación Q entre los conjuntos V y W , definimos la composición de estas dos relaciones, y la denotamos como $R \circ Q$, como una nueva relación entre los conjuntos U y W . Un ejemplo de esta composición lo encontramos en la sentencia “busco hotel cerca de alguna playa alejada de un centro urbano”, donde la relación R se corresponde con la expresión “hotel cerca de alguna playa”, y la relación Q se corresponde con la expresión “playa alejada de un centro urbano”. La composición entre R y Q define una relación entre $u \in U$ y $w \in W$ a partir de la relación que mantienen tanto u como w con cada uno de los elementos de V . Parece razonable que, dados tres elementos u , v y w , la relación que existe entre u y w no sea mejor que la peor de las relaciones que hay entre u y v y entre v y w , y dados dos elementos u y w , la relación que existe entre ellos debe ser la de aquel v con quien están mejor relacionados. Con esto en mente, Zadeh propone modelar este vínculo mediante la siguiente expresión:

$$\mu_{R \circ Q}(u, w) = \sup_v \min\{\mu_R(u, v), \mu_Q(v, w)\}$$

Esta expresión se conoce como composición sup-mín, y generaliza la composición de relaciones precisas, según la cual u está relacionada con w si y sólo si existe un v tal que u está relacionada con v y v con w . También podemos sustituir en esta expresión el mínimo por el producto o, en general, por cualquier t-norma. Cada una de las t-normas proporciona propiedades particulares a la composición de relaciones. Por ejemplo, si descomponemos cada una de las relaciones en α -cortes, entonces la igualdad $(R \circ Q)_\alpha = R_\alpha \circ Q_\alpha$ sólo se satisface si la composición es sup-mín.

Por otra parte, es fácil ver que la composición de relaciones se puede formular en términos de proyección y extensión cilíndrica de relaciones. En general, a partir de dos relaciones R y Q , definidas respectivamente en $U_1 \times \dots \times U_r$ y $U_q \times \dots \times U_n$, donde $q \leq r$, definimos su composición como $R \circ Q = Pr(Ex(R) \cap Ex(Q); U_1 \times \dots \times U_{q-1} \times U_{r+1} \times \dots \times U_n)$.

La frecuencia con la que aparecen relaciones binarias en el lenguaje ha motivado un estudio más detallado de las propiedades de su composición, que mostramos aquí brevemente. Supongamos las relaciones borrosas R en $U \times V$, Q en $V \times W$, S en $V \times W$ y T en $W \times X$. Se puede probar entonces el siguiente conjunto de propiedades:

- R1. *Asociativa*: $R \circ (Q \circ T) = (R \circ Q) \circ T$.
- R2. *Distributiva* respecto a la unión: $R \circ (Q \cup S) = (R \circ Q) \cup (R \circ S)$.
- R3. *Distributiva débil* respecto a la intersección: $R \circ (Q \cap S) \subseteq (R \circ Q) \cap (R \circ S)$.
- R4. *Monotonía*: Si $Q \subseteq S$ entonces $R \circ Q \subseteq R \circ S$.

Un caso particular de uso de la composición de relaciones nos permite calcular la imagen de un determinado conjunto borroso $A \subset U$ a través de una relación $R \subset U \times V$. La forma que adopta este cálculo es la siguiente:

$$\mu_{A \circ R}(v) = \sup_{u \in U} \min\{\mu_A(u), \mu_R(u, v)\}$$

Decimos que A induce, a través de R , el conjunto borroso $B = A \circ R$. Esto supone una generalización de un razonamiento muy común en el mundo de lo preciso: si $x = a$ e $y = f(x)$, entonces $y = f(a)$. En términos de proyección y extensión cilíndrica $B = Pr(Ex(A) \cap R; V)$ (véase la Figura 7.9).

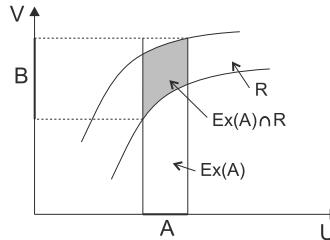


Figura 7.9: Representación de la composición en términos de proyección y extensión cilíndrica.

Supongamos que A es un número borroso resultado de representar el término lingüístico “las cuatro” utilizado en un sentido aproximado. Ahora supongamos que R es una relación del tipo de “mucho después”. El conjunto borroso $A \circ R$ representa el conjunto de números que son mucho mayores que el número A , tal y como se puede ver en el ejemplo de la Figura 7.10.

Si $\mu_R(u, v) = 1$ para $u \geq v$ y $\mu_R(u, v) = 0$ en cualquier otro caso, la expresión de la composición se reduce a:

$$\mu_{A \circ R}(u) = \mu_{[A, +\infty)}(u) = \sup_{v \leq u} \mu_A(v)$$



Figura 7.10: Un ejemplo de composición borrosa en el dominio de los números reales.

La composición de relaciones amplía las posibilidades de los conjuntos borrosos como herramienta para modelar el significado de las sentencias del lenguaje natural. Como ejemplo, supongamos un sistema informático de reservas que admite búsquedas mediante sentencias del tipo “Busco hotel económico cerca de alguna bonita playa alejada de un centro urbano”. La expresión S = “hotel económico” se puede modelar mediante un conjunto borroso $E \subset H$, cuya función de pertenencia $\mu_E(h)$, para cualquier $h \in H$, nos proporciona una medida de compatibilidad de cada hotel del conjunto H con la definición de hotel económico. La expresión “cerca de alguna bonita playa” es equivalente a “cerca de alguna playa y siendo esta playa bonita”, lo que, como conjunción, podemos modelar mediante la combinación de la relación borrosa $C \subset H \times P$, cuya función de pertenencia $\mu_C(h, p)$ nos proporciona una medida del grado de satisfacción de la relación cerca, para cada pareja $(h, p) \in H \times P$; y el conjunto borroso $B \subset P$, cuya función de pertenencia $\mu_B(p)$, nos proporciona una medida de compatibilidad de cada playa del conjunto P con la definición de playa bonita. La expresión “alejada de un centro urbano” la podemos modelar mediante una relación borrosa $A \subset P \times U$, cuya función de pertenencia $\mu_A(p, u)$ asigna un grado de satisfacción de la relación alejada, para cada pareja $(p, u) \in P \times U$. Modelar la sentencia original supone realizar la combinación y composición de cada una de las expresiones que la componen, con el fin de encontrar una función de pertenencia $\mu_S(h, u)$ para la relación borrosa S que determina el grado de compatibilidad de cada pareja $(h, u) \in H \times U$ con los criterios que hemos manifestado para el hotel que buscamos:

$$S = Ex(E) \cap (C \cap Ex(B)) \circ A$$

$Ex(B) \subset H \times P$ nos permite realizar la combinación de B con la relación C para modelar la expresión “un hotel cerca de alguna bonita playa”. La composición $(C \cap Ex(B)) \circ A$ modela la expresión “un hotel cerca de alguna playa alejada de un centro urbano”. $Ex(E) \subset H \times U$ nos permite realizar la combinación de E con la composición anterior, lo que modela finalmente la sentencia original S , y da lugar al siguiente cálculo:

$$\mu_S(h, u) = \min\{\mu_E(h), \sup_{p \in P} \min\{\mu_C(h, p), \mu_A(p, u), \mu_B(p)\}\}$$

A partir de esta relación, podemos preguntar a nuestro sistema de reservas por algún hotel de estas características para el que pedimos adicionalmente que el centro urbano sea de “aproximadamente más de mil habitantes”. Esta restricción para el tamaño de

la población de la que deseamos alejarnos se puede modelar mediante un conjunto borroso $M \subset U$, cuya función de pertenencia $\mu_M(u)$ calcula la compatibilidad entre el tamaño de una población y la restricción que imponemos. Podemos añadir M a nuestro cálculo de dos maneras:

- Mediante la combinación de dos restricciones sobre la relación entre playas y centros urbanos, para lo que realizamos la extensión cilíndrica de M en $P \times U$, de modo que obtenemos $S \cap Ex(M)$. Disponemos así de una nueva relación $BH \subset H \times U$ entre hoteles y centros urbanos, cuyo grado de satisfacción viene dado por la expresión:

$$\mu_{BH}(h, u) = \min\{\mu_S(h, u), \mu_M(u)\}$$

- Mediante una operación de proyección que permitirá obtener, para cada hotel $h \in H$, su grado de compatibilidad con todos los criterios establecidos. Obtenemos, por tanto, un conjunto borroso $BH \subset H$ que permite ordenar el conjunto de hoteles según la expresión:

$$\mu_{BH}(h) = \sup_{u \in U} \min\{\mu_S(h, u), \mu_M(u)\}$$

7.8 El condicional

Un tipo de predicado de especial relevancia en la representación del conocimiento es el condicional, y aunque aparece en el lenguaje de múltiples maneras, como son: “no pienso salir hasta que te vayas”, o “avísame en caso de que salgas”, se suele estudiar bajo la forma común “si ... entonces ...”. Una expresión de la forma “Si x es A entonces y es B ” puede adoptar múltiples significados en el lenguaje [Copi, 1999]. El primer intento de formalizar el significado del condicional aparece ligado al cálculo proposicional, donde se denota como $A \rightarrow B$, y el símbolo ‘ \rightarrow ’ representa una relación entre predicados conocida como *implicación*. En el caso de predicados A y B precisos la implicación se suele modelar como $\neg A \vee B$ (no A o B), lo que se conoce como implicación material. Como primera aproximación a la representación del condicional, nos interesa encontrar una implicación multivaluada, extensión de la implicación material clásica, y que permita modelar expresiones condicionales en las que los predicados A y B sean borrosos, como puede ser el caso de “si encuentras el agua muy caliente, añade agua fría”. Esta implicación es un caso particular de relación borrosa que denotamos como:

$$\mu_{A \rightarrow B}(u, v) = I(\mu_A(u), \mu_B(v))$$

donde $I : [0, 1] \times [0, 1] \rightarrow [0, 1]$. Podemos obtener diversas implicaciones, sin más que expresar la fórmula clásica $\neg A \vee B$ mediante el uso de una t-conorma S y una negación fuerte N , según la expresión:

$$I(a, b) = S(n(a), b)$$

Sustituyendo las distintas t-conormas, y la negación fuerte presentadas en la sección 7.4, obtenemos las siguientes expresiones para la relación de implicación:

$$\begin{aligned}\mu_{A \rightarrow B}(u, v) &= \max\{1 - \mu_A(u), \mu_B(v)\} \\ \mu_{A \rightarrow B}(u, v) &= 1 - \mu_A(u) + \mu_A(u)\mu_B(v) \\ \mu_{A \rightarrow B}(u, v) &= \min\{1 - \mu_A(u) + \mu_B(v), 1\}\end{aligned}$$

Esta forma de obtener la relación de implicación recibe el nombre de *S-implicación*, y debemos en gran medida su caracterización a Trillas [Trillas y Valverde, 1985]. Esta caracterización se expresa mediante un conjunto de propiedades que cabe esperar de una relación de implicación:

I1. $I(0, 0) = I(0, 1) = I(1, 1) = 1, I(1, 0) = 0.$

Esta propiedad nos dice que toda implicación borrosa ha de recuperar el resultado clásico en el caso de manejar predicados precisos.

I2. Si $a \leq a'$ entonces $I(a, b) \geq I(a', b)$, para cualquier $b \in [0, 1]$.

Un incremento en la verdad del antecedente no conduce a un incremento en la verdad de la implicación.

I3. Si $b \leq b'$ entonces $I(a, b) \leq I(a, b')$, para cualquier $a \in [0, 1]$.

Un incremento en la verdad del consecuente conduce a un incremento en la verdad de la implicación.

I4. $I(0, b) = 1$, para cualquier $b \in [0, 1]$.

Si el antecedente es completamente falso, la implicación es completamente cierta, independientemente del valor de verdad del consecuente.

I5. $I(a, 1) = 1$, para cualquier $a \in [0, 1]$.

Si el consecuente es cierto, la implicación es completamente cierta, independientemente del valor de verdad del antecedente.

I6. $I(1, b) = b$, para cualquier $b \in [0, 1]$.

Si el antecedente es cierto, la implicación es tan cierta como lo es su consecuente.

I7. $I(a, I(b, c)) = I(b, I(a, c))$, para cualesquiera $a, b, c \in [0, 1]$.

Recoge el llamado principio de intercambio, que nos dice que $(A \rightarrow (B \rightarrow C)) \Leftrightarrow ((A \wedge B) \rightarrow C) \Leftrightarrow (B \rightarrow (A \rightarrow C))$.

I8. $I(a, b) \geq b$, para cualesquiera $a, b \in [0, 1]$.

Es la forma de representar numéricamente la fórmula $B \rightarrow (A \rightarrow B)$.

I9. $I(a, a) = 1$, para cualquier $a \in [0, 1]$.

Nos permite generalizar la tautología $A \rightarrow A$.

I10. $I(a, b) = 1$ si y sólo si $a \leq b$, para cualquier $a, b \in [0, 1]$.

La implicación es completamente cierta en tanto que el consecuente es al menos tan cierto como el antecedente. Por otra parte, expresa que la implicación define un orden.

I11. $I(a, b) = I(n(b), n(a))$, para cualquier $a, b \in [0, 1]$, y n una negación fuerte.

Representa la ley de contraposición, que nos dice que $(A \rightarrow B) \Leftrightarrow (\neg B \rightarrow \neg A)$.

I12. I es una función continua.

Que nos asegura que la implicación no se comporta bruscamente ante pequeños cambios en el valor de verdad del antecedente o del consecuente.

Estas propiedades no son independientes entre sí. En particular, las propiedades 4 y 5 son consecuencia de la 1, 2 y 3. Además, una implicación I que satisface las propiedades 3, 7 y 10 también satisface las propiedades 1, 2, 4, 5, 6, 8 y 9.

Pero podríamos preguntarnos si esta extensión de la fórmula clásica es la única manera de modelar el condicional cuando los predicados no son precisos. Al fin y al cabo, es fácil pensar en la expresión clásica como en una simplificación del significado de consecuencia lógica en el caso particular de manejar predicados precisos. Podríamos pensar en otras posibilidades para definir una relación de implicación, por ejemplo, utilizando la siguiente igualdad de la teoría de conjuntos:

$$\overline{A} \cup B = \overline{A - B} = \{Z \mid A \cap Z \subseteq B\}$$

donde el símbolo $\overline{-}$ denota la diferencia de conjuntos. Esta igualdad nos permite introducir un nuevo tipo de implicación, que responde a la siguiente expresión:

$$I(a, b) = \sup\{c \in [0, 1] \mid T(a, c) \leq b\}$$

donde T hace referencia a una t-norma. Esta forma de implicación responde al nombre de *R-implicación*, ya que la expresión de la que se obtiene guarda relación con el concepto de residuo en el álgebra [Goguen, 1969].

Un último tipo de implicación que encontramos en la bibliografía trata de modelar aquellos condicionales de la forma “si ... entonces ... si no ...”, y lo hace mediante la expresión:

$$I(a, b) = S(n(a), T(a, b))$$

donde S es una t-conorma, n es una negación fuerte, y T es la t-norma n-dual de S , obtenida mediante la expresión $T(a, b) = n(S(n(a), n(b)))$. Esta forma de implicación recibe el nombre de *QL-implicación*, ya que se corresponde con la lógica que subyace en los predicados de la mecánica cuántica. Si consideramos el condicional siguiente:

$$\begin{aligned} &\text{Si } x \in A \text{ entonces } y \in B, \\ &\text{si } x \notin A \text{ entonces } y \in C \end{aligned}$$

se puede interpretar como una restricción sobre el valor de (x, y) , de modo que $(x, y) \in (A \times B)$, o bien $(x, y) \in (\overline{A} \times C)$, lo que se expresa como:

$$\mu_R(x, y) = S(T(\mu_A(x), \mu_B(y)), T(n(\mu_A(x)), \mu_C(y)))$$

Si C es desconocido, entonces $\mu_C(y) = 1$ para cualquier $y \in [0, 1]$, y recuperamos la expresión de la QL-implicación. Si C no está definido, entonces $\mu_C(y) = 0$ para cualquier $y \in [0, 1]$, y obtenemos una interpretación del condicional que no se corresponde con una implicación, sino con una conjunción $\mu_R(x, y) = T(\mu_A(x), \mu_B(y))$. Esta interpretación del condicional es importante, ya que está presente en múltiples aplicaciones de control borroso. En estas aplicaciones un conjunto de condicionales de la forma “si x es A entonces y es B ” permite describir una relación borrosa R entre x e y , satisfaciendo que $A \times B \subseteq R$. Para cada uno de los condicionales, su antecedente representa una observación, mientras el consecuente representa una acción de control, como es el caso de “si la temperatura es muy alta añadir abundante agua fría” [Dubois y Prade, 1991].

Como conclusión, podemos decir que un predicado condicional de la forma “Si x es A entonces y es B ” se puede modelar mediante una función $J : [0, 1] \times [0, 1] \rightarrow [0, 1]$ que debe proporcionar una significación para cada particular uso del predicado en el lenguaje. Aquí nos hemos limitado a exponer algunos de estos usos, muy habituales en las aplicaciones de los conjuntos borrosos en la ingeniería, y que interpretan el condicional como una implicación I o una t-norma T .

En la Tabla 7.2 mostramos como resumen algunas de las implicaciones de uso común, junto con las propiedades que satisfacen. Como vemos, la implicación de Lukasiewicz es la que satisface un mayor número de propiedades, siendo la única R-implicación que también es S-implicación.

Nombre	Forma	Tipo	Propiedades
Kleene-Dienes	$\max(1 - a, b)$	S-implicación ($S = \max$) QL-implicación ($S = \min(1, a + b)$)	I1-I8,I11-I12
Reichenbach	$1 - a + ab$	S-implicación ($S = a + b - ab$)	I1-I8,I11-I12
Lukasiewicz	$\min(1, 1 - a + b)$	S-implicación ($S = \min(1, a + b)$) R-implicación ($T = \max(0, a + b - 1)$)	I1-I12
Gödel	$1 \text{ si } a \leq b$ $b \text{ si } a > b$	R-implicación ($T = \min$)	I1-I10
Goguen	$1 \quad \text{si } a = 0$ $\max(1, b/a) \quad \text{si } a \neq 0$	R-implicación ($T = \text{producto}$)	I1-I10,I12
Zadeh	$\max(1 - a, \min(a, b))$	QL-implicación ($S = \max$)	I1,I3,I6,I12

Tabla 7.2: Algunas implicaciones multivaluadas.

7.9 Cualificación lingüística

De cualquier sentencia del lenguaje podemos decir si nos parece cierta o falsa. Y podemos añadir el uso de modificadores lingüísticos para dar lugar a apreciaciones del tipo de “muy cierta”, “más o menos cierta” o “bastante falsa”. A estas expresiones les podemos asignar un significado que viene determinado por un conjunto borroso en el intervalo $[0, 1]$, y que por su sentido se corresponde con un valor de verdad borroso.

En la sección 7.4 exponíamos que la noción de significado tiene una validez local, restringida a un uso particular del lenguaje. Lo mismo cabe decir de la noción de verdad. Bellman y Zadeh argumentan que los conjuntos borrosos nos permiten modelar aquella parte del lenguaje formada por sentencias que no son universalmente

ciertas, y cuya certeza se define de un modo local, con respecto a alguna sentencia de referencia cuya certeza es segura [Bellman y Zadeh, 1977].

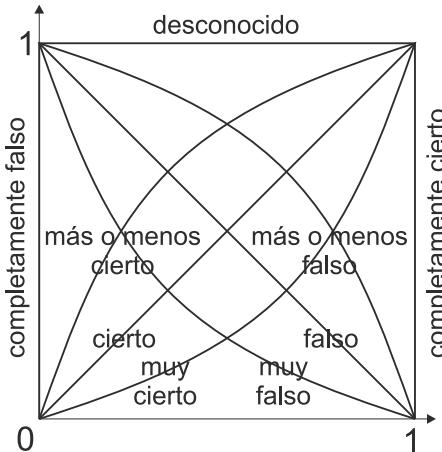


Figura 7.11: Representación borrosa de distintos valores de verdad expresados de un modo lingüístico.

Supongamos la sentencia “X es A es τ -cierto”, donde τ es un valor de verdad borroso. Bellman y Zadeh proponen construir aquella sentencia cierta que subyace en ella, mediante la siguiente equivalencia:

$$\text{“X es A es } \tau\text{-cierto”} \Leftrightarrow \exists B, \text{ “X es B es cierto” y } \mu_B = f(\mu_\tau, \mu_A)$$

Queremos definir el grado de verdad de la sentencia “X es A” sabiendo que es cierto que “X es B”. En el caso particular en que $B = \{u_0\}$, dicho grado de verdad es simplemente $\mu_A(u_0)$. En el caso borroso, recuperamos la expresión de la compatibilidad $\mu_{C(A,B)} = \mu_A(B)$, y obtenemos un conjunto borroso en $[0, 1]$, mediante la expresión:

$$\mu_\tau(v) = \mu_{C(A,B)}(v) = \sup_{u: v=\mu_A(u)} \mu_B(u), \quad v \in [0, 1]$$

El problema al que nos enfrentamos es el de obtener B a partir de la sentencia “X es A es τ -cierto”, esto es, a partir de los conjuntos borrosos A y τ . Cualquier B que sea solución de la ecuación de compatibilidad nos sirve. La solución más grande es la siguiente:

$$\mu_B = \mu_\tau \circ \mu_A$$

esto es, $\mu_B(u) = \mu_\tau(\mu_A(u))$. En esta forma, un conjunto borroso τ en el intervalo $[0, 1]$ tiene una interpretación en términos de valores de verdad. Así, la expresión “X es A es cierto” es equivalente a “X es A”, luego podemos decir que el conjunto borroso $\mu_\tau(v) = v$, para cualquier $v \in [0, 1]$ representa el significado del valor de verdad ‘cierto’. La expresión “X es A es falso” es equivalente a “X es $\neg A$ ”, con $\mu_{\neg A} = 1 - \mu_A$,

luego podemos decir que el conjunto borroso $\mu_\tau(v) = 1 - v$ representa el significado del valor de verdad ‘falso’. La expresión “X es A es desconocido” es equivalente a “X es U”, luego podemos decir que el conjunto borroso $\mu_\tau(v) = 1$ representa el valor de verdad ‘desconocido’. A este esquema podemos añadir el uso de modificadores, tal y como muestra la Figura 7.11.

Si A y B son conjuntos precisos, recuperamos los valores de verdad 1 y 0: si $B \subseteq A$ entonces $\mu_\tau = \{1\}$, y si $B \subseteq \overline{A}$ entonces $\mu_\tau = \{0\}$. Por otra parte, la expresión $\mu_B(u) = \mu_\tau(\mu_A(u))$ no es aplicable cuando A es un conjunto borroso y τ es un valor de verdad preciso $\tau = v$, siendo $v \in]0, 1[$, ya que nos encontraríamos con la paradoja de obtener un valor preciso a partir de una sentencia borrosa cualificada con un valor de verdad parcial.

7.10 Razonamiento borroso

Hasta ahora hemos abordado el estudio de los conjuntos borrosos desde la perspectiva de su capacidad para representar aquel conocimiento que expresamos mediante el lenguaje natural. Nuestra preocupación se va a centrar ahora en la forma que adopta el razonamiento cuando involucra predicados vagos. Si la lógica clásica se ocupa del análisis de aquellas oraciones que pueden ser verdaderas o falsas, necesitamos una *lógica borrosa* que se ocupe del análisis de las oraciones cuya verdad sea una cuestión de grado.

Una de las reglas de inferencia básicas en la lógica clásica es el *Modus Ponens*, ya que permite la deducción de nuevas observaciones y condiciones a partir de las observaciones y condiciones disponibles, y que podemos reducir al siguiente esquema:

CONDICIÓN:	Si X es A entonces Y es B
OBSERVACIÓN:	X es A
CONCLUSIÓN:	Y es B

donde $A \subset U$ y $B \subset V$ son dos conjuntos precisos. Podemos interpretar esta regla de inferencia como la conjunción $Ex(A) \cap (A \rightarrow B)$ del hecho y la condición, y su posterior proyección en V $B = Pr(Ex(A) \cap (A \rightarrow B); V)$. Dicho de otro modo, podemos entender esta regla de inferencia como la composición de una relación condicional $A \rightarrow B$ con el conjunto A , lo que permite inducir el conjunto B como $B = A \circ (A \rightarrow B)$, es decir, la imagen de A a través de la relación $A \rightarrow B$. Esta composición se conoce como *regla composicional de inferencia* [Zadeh, 1973] y su cálculo obedece a la siguiente expresión:

$$\mu_B(v) = \sup_{u \in U} T(\mu_A(u), J(\mu_A(u), \mu_B(v)))$$

donde T es, en principio, cualquier t-norma. Ahora bien, esta expresión de la regla composicional de inferencia también es aplicable cuando tanto A como B son conjuntos borrosos, por lo que nos permite generalizar el *Modus Ponens* a la manipulación de predicados vagos. Con una salvedad, y es que no está garantizado que el resultado

de la composición $A \circ (A \rightarrow B)$ sea B . Por otra parte, nada nos impide utilizar esta expresión cuando la observación A' no coincide con el antecedente A , en cuyo caso, la diferencia entre ambos se propaga hacia la conclusión B' , que se obtiene como $B' = A' \circ (A \rightarrow B)$. Obtenemos así la forma general de la regla composicional de inferencia:

$$\mu_{B'}(v) = \sup_{u \in U} T(\mu_{A'}(u), J(\mu_A(u), \mu_B(v)))$$

Tenemos, por tanto, un nuevo esquema de razonamiento que llamamos *Modus Ponens Generalizado* y cuya forma es la siguiente:

$$\frac{\begin{array}{c} \text{CONDICIÓN: } \text{Si } X \text{ es } A \text{ entonces } Y \text{ es } B \\ \text{OBSERVACIÓN: } X \text{ es } A' \\ \hline \text{CONCLUSIÓN: } Y \text{ es } B' \end{array}}{} \quad Y \text{ es } B'$$

Podemos utilizar esta regla de inferencia en aquellos problemas para los que disponemos de conocimiento en forma de sentencias condicionales, del tipo de “si la temperatura es alta entonces la presión desciende”, y observamos algo parecido a “la temperatura es moderada”. La regla de inferencia nos permite obtener una conclusión, y la expresa en forma matemática. Con todo, la regla composicional de inferencia nos permite utilizar cualquier combinación formada por una t-norma y un condicional. Cada una de estas combinaciones supone obtener una conclusión diferente tras la aplicación de la regla, y por tanto, cabe preguntarse si cualquiera de estas combinaciones permite obtener una conclusión razonable. Para ilustrar lo dicho introduciremos otra regla de inferencia bien conocida, el *Modus Tollens*, cuya forma es la siguiente:

$$\frac{\begin{array}{c} \text{CONDICIÓN: } \text{Si } X \text{ es } A \text{ entonces } Y \text{ es } B \\ \text{OBSERVACIÓN: } Y \text{ es } \neg B \\ \hline \text{CONCLUSIÓN: } X \text{ es } \neg A \end{array}}{} \quad X \text{ es } \neg A$$

Si utilizamos una implicación que satisface la ley de contraposición, esto es, $(A \rightarrow B) \Leftrightarrow (\neg B \rightarrow \neg A)$, entonces el *Modus Ponens* y el *Modus Tollens* son equivalentes. Sin embargo, implicaciones como la de Gödel o la de Goguen, y en general, todas las R-implicaciones excepto la de Lukasiewicz, no satisfacen la ley de contraposición. Esto no significa que estas implicaciones conduzcan a conclusiones poco razonables. Todo depende de las propiedades que exigimos a los razonamientos que realizamos en cada uso del lenguaje.

Debemos, por consiguiente, estudiar el conjunto de propiedades que cabe pedir al *Modus Ponens Generalizado*. Estas propiedades condicionarán la selección de aquellas parejas de t-normas y condicionales que podrán formar parte de la regla composicional de inferencia. Algunas de estas propiedades pueden ser las siguientes:

MP1. $A' = A \Rightarrow B' = B$.

Que exige la coincidencia con el *Modus Ponens* clásico cuando la observación coincide con el antecedente de la condición.

MP2. $B \subseteq B'$.

No es posible inferir un resultado B' más preciso que el consecuente B de la condición, siempre y cuando A' esté normalizado.

MP3. $A'' \subseteq A' \Rightarrow B'' \subseteq B'$.

O propiedad de monotonía, según la cual cuanto más precisa es la observación, más precisa es la conclusión.

MP4. $A' = \overline{A} \Rightarrow B' = V$.

Si la observación es $\neg A$, la condición no es aplicable, y por tanto ignoramos el valor que toma la variable Y . Se exige que \overline{A} esté normalizado.

MP5. $B' = A' \circ (A \rightarrow B) \Rightarrow \overline{A'} = \overline{B'} \circ (A \rightarrow B)$.

Esta propiedad generaliza la simetría que hay entre el *Modus Ponens* ($A \wedge (A \rightarrow B) \Rightarrow B$) y el *Modus Tollens* ($\neg B \wedge (A \rightarrow B) \Rightarrow \neg A$).

MP6. $B \subset B'$ si $A' \not\subseteq A$.

Versión más restrictiva de MP2.

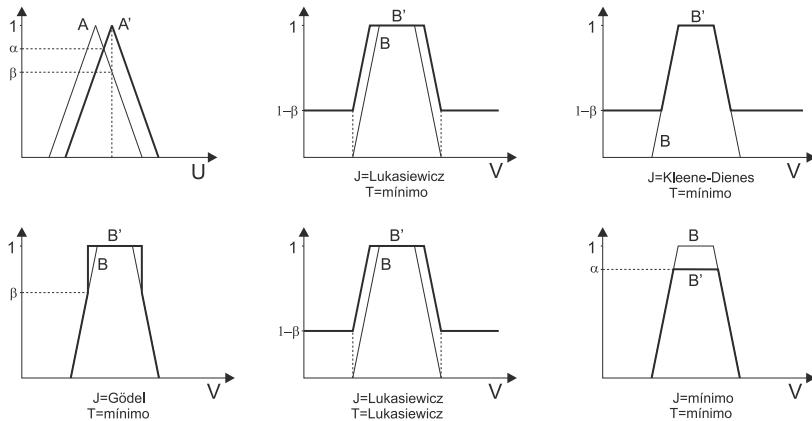
MP7. $A' \subseteq A \Rightarrow B' = B$.

Si la observación es más precisa que el antecedente del condicional, la conclusión obtenida es el propio consecuente (suponemos que A' está normalizada). Supongamos que $A' = \text{MUY } A$, donde modelamos el modificador ‘MUY’ mediante una función de modificación. Esta propiedad nos dice que la conclusión obtenida es B . Algunos autores sostienen que se debería concluir MUY B , apoyándose en ejemplos del lenguaje natural como el del condicional “si los tomates están rojos entonces están maduros” ante la observación “los tomates están muy rojos” [Fukami y otros, 1980]. Según estos autores, la conclusión natural es “los tomates están muy maduros”. El problema es que esta conclusión entra en contradicción con la propiedad MP2. Se podría argumentar que un razonamiento de estas características responde a una interpretación del condicional como relación funcional entre los grados de color y madurez, más que al significado propio de una implicación.

En la bibliografía podemos encontrar un estudio completo de las distintas combinaciones entre t-normas y condicionales, y las propiedades que satisface el *Modus Ponens Generalizado* a que dan lugar [Mizumoto y Zimmermann, 1982]. Un estudio de esas características excede el espacio y las intenciones de un capítulo introductorio a los conjuntos borrosos, por lo que nos limitaremos a resumir (véase la Tabla 7.3) aquellas propiedades que satisfacen algunas de estas combinaciones.

Como ejemplo de aplicación del *Modus Ponens Generalizado*, mostramos en la Figura 7.12 las distintas conclusiones obtenidas a partir de la regla “Si X es A entonces Y es B ” tras la observación “ X es A' ”.

t-norma	Condicional	Propiedades
$T_Z(a, b) = \min(a, b)$	Lukasiewicz ($\min(1, 1 - a + b)$)	MP2-4
$T_G(a, b) = ab$		MP2-4
$T_L(a, b) = \max(a + b - 1, 0)$		MP1-4,7
$T_Z(a, b) = \min(a, b)$	Kleene-Dienes ($\max(1 - a, b)$)	MP2-4
$T_G(a, b) = ab$		MP2-4
$T_L(a, b) = \max(a + b - 1, 0)$		MP1-4,7
$T_Z(a, b) = \min(a, b)$	Gödel (1 si $a \leq b$, 0 si $a > b$)	MP1-4,6-7
$T_G(a, b) = ab$		MP1-4
$T_L(a, b) = \max(a + b - 1, 0)$		MP1-4,7
$T_Z(a, b) = \min(a, b)$	Mínimo ($\min(a, b)$)	MP1,3,7
$T_G(a, b) = ab$		MP1,3,7
$T_L(a, b) = \max(a + b - 1, 0)$		MP1,3,7

 Tabla 7.3: Algunos pares de t-normas y condicionales en el *Modus Ponens*.

 Figura 7.12: Un ejemplo de inferencia realizada mediante la aplicación del *Modus Ponens Generalizado* utilizando distintas parejas de condicionales y t-normas.

7.10.1 Condicionales con antecedente múltiple

Fácilmente podemos extender esta regla de inferencia al caso en el que el antecedente es múltiple, cuya forma es la siguiente:

CONDICIÓN: Si X_1 es A_1 y X_2 es A_2 entonces Y es B

OBSERVACIÓN: X_1 es A'_1 y X_2 es A'_2

CONCLUSIÓN:

Y es B'

donde $A_1 \subset U_1$ y $A_2 \subset U_2$. Si realizamos la extensión cilíndrica de A_1 y A_2 a $U_1 \times U_2$ podemos modelar la conjunción “ X_1 es A_1 y X_2 es A_2 ” mediante la combinación $Ex(A_1) \cap Ex(A_2)$.

Hay que tener en cuenta que estamos asumiendo que las variables X e Y no guardan ninguna relación entre sí. En caso contrario, no daríamos un significado correcto para la sentencia anterior. La regla composicional de inferencia adopta la siguiente expresión:

$$\mu_{B'}(v) = \sup_{u_1, u_2} T(\mu_{A'_1}(u_1), \mu_{A'_2}(u_2), J(T(\mu_{A_1}(u_1), \mu_{A_2}(u_2)), \mu_B(v)))$$

Esta forma del condicional resulta de especial interés en el desarrollo de aplicaciones basadas en conjuntos borrosos. Estas aplicaciones tienen en común el realizar una representación explícita del conocimiento necesario para resolver un problema. Dicha representación se realiza mediante un conjunto de reglas condicionales, agrupadas en lo que se conoce como *base de reglas*.

Esta aproximación es la seguida en el diseño de sistemas de control borroso, tal y como hemos anticipado en la sección 7.8. El control borroso trata de sintetizar en una base de reglas la experiencia de un operador humano a cargo del control de un proceso. El conjunto de reglas resume las distintas situaciones en que se puede encontrar el proceso, y las acciones que en consecuencia se han de realizar para su control. La capacidad del *Modus Ponens Generalizado* para razonar en aquellas condiciones en las que las observaciones no coinciden exactamente con el antecedente de las reglas, permiten reducir el número de reglas necesarias para un control efectivo, ya que el razonamiento borroso realiza una interpolación entre las distintas situaciones representadas en la base de reglas. El capítulo 14 examina con más detalle los principios del diseño de sistemas de control borroso.

Otro ejemplo paradigmático de uso de los conjuntos borrosos lo encontramos en el diseño de sistemas de clasificación borrosa. En un problema de clasificación un objeto determinado x , descrito mediante un conjunto de propiedades $x = (x_1, \dots, x_n)$, ha de ser asignado a una de las clases del conjunto $C = \{c_1, \dots, c_m\}$. Un clasificador borroso κ se define como una aplicación $\kappa : \mathbb{R}^n \rightarrow C$, que obtiene los valores de un conjunto de funciones discriminantes $\mu_{c_1}(x), \dots, \mu_{c_m}(x)$, representadas mediante funciones de pertenencia, y asigna x a aquella clase cuya función discriminante devuelve un valor mayor. Una realización de un clasificador borroso mediante el uso de condicionales permite modelar las fronteras entre las distintas clases mediante conjuntos borrosos asociados a etiquetas lingüísticas (alto, normal, etc). Algunos resultados bien conocidos demuestran que sistemas clasificadores basados en condicionales borrosos pueden aproximar con una precisión arbitraria cualquier función continua $f : \mathbb{R}^n \rightarrow \mathbb{R}$ en un compacto $U \subset \mathbb{R}^n$ [Castro y Delgado, 1996; Kuncheva, 2000]. Tenemos pues, que tal clasificador borroso se comporta como un aproximador universal. Basta con un adecuado diseño de la base de reglas.

7.11 Cuantificación

Como hemos visto en el capítulo 2, la lógica de predicados de primer orden introduce dos cuantificadores presentes en el lenguaje: el cuantificador universal, representado matemáticamente por el símbolo \forall , y el cuantificador existencial, representado por \exists .

Sin embargo, el lenguaje natural admite una variedad de formas de cuantificar mucho más amplia, recorriendo múltiples situaciones intermedias entre los cuantificadores universal y existencial: mucho, pocos, bastante, la mayoría, casi, etc.

Pensemos en sentencias comunes del tipo de “aproximadamente cinco personas miden más de 1'80m” o “la mayoría del congreso rechazó la moción”. En estas sentencias el hablante cuantifica el número (“aproximadamente cinco”) o proporción (“la gran mayoría”) de individuos que satisfacen sendos predicados precisos (“miden más de 1'90m”, “rechazó la moción”), y lo hace de una forma que no es obviamente universal (“todos”) ni existencial (“al menos uno”), pero que indudablemente todos sabemos evaluar. Por otra parte, esta forma de cuantificar es inherentemente imprecisa, lo que nos sugiere realizar una definición borrosa de los cuantificadores que aparecen en el lenguaje, y que pasamos a llamar *cuantificadores borrosos* (véase la Figura 7.13).

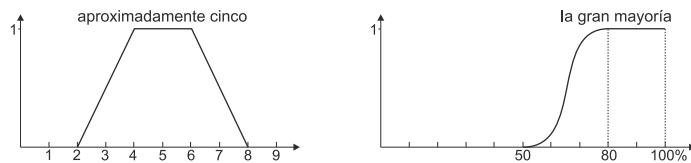


Figura 7.13: Representación gráfica de las funciones de pertenencia correspondientes a los cuantificadores “aproximadamente cinco” y “la gran mayoría”.

La primera propuesta para modelar cuantificadores mediante conjuntos borrosos se la debemos a Zadeh, que aporta una solución sencilla [Zadeh, 1983]. Considera dos tipos básicos de cuantificadores:

- *Cuantificadores absolutos*, aquéllos que representan cantidades, y se pueden modelar mediante números borrosos definidos en dominios como \mathbb{N} o \mathbb{R}^+ . Ejemplos de cuantificadores absolutos son “aproximadamente cinco”, “algún”, etc.
- *Cuantificadores relativos*, aquéllos que representan proporciones, y se pueden modelar mediante conjuntos borrosos en $[0, 1]$. Ejemplos de cuantificadores relativos son “la gran mayoría”, “pocos”, “muchos”, etc.

Zadeh propone el siguiente mecanismo en dos pasos para evaluar sentencias, y así obtener el grado de verificación de un predicado que contiene cuantificadores borrosos:

1. Evaluar el número de elementos (“la gran mayoría”), respecto a un conjunto de referencia (el total de diputados), que satisfacen el predicado (“rechazó la moción”), es decir, su cardinalidad.
2. Calcular la medida en que la cardinalidad es compatible con el cuantificador borroso.

En general, para una sentencia *unitaria* (con una propiedad) del tipo “ Q son A ”, definida sobre un conjunto de referencia E , podemos definir el cuantificador Q mediante

un número borroso cuya función de pertenencia es μ_Q , de modo que la verificación del predicado viene dada por la expresión:

$$g = \mu_Q(\text{card}_E(A))$$

donde card_E es una función que calcula la cardinalidad, y que adopta una forma diferente según sea el cuantificador absoluto, o relativo:

$$\text{(absoluto)} \quad \text{card}_E(A) = \sum_{e \in E} \mu_A(e) \quad \text{(relativo)} \quad \text{card}_E(A) = \frac{\sum_{e \in E} \mu_A(e)}{|E|}$$

Ejemplo 7.1. Consideremos la evaluación de la sentencia “aproximadamente cinco personas miden más de 1'80m” sobre un conjunto de referencia formado por diez individuos $E=\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$, donde la estatura de cada uno de ellos viene dada por el conjunto Estaturas= $\{1'90, 1'70, 1'65, 1'81, 1'80, 1'79, 1'75, 1'55, 1'81, 1'82\}$. Podemos representar gráficamente la satisfacción del predicado preciso $M=“mide más de 1'80m”$, para cada uno de los elementos e_i del conjunto de referencia E , tal y como se indica en la Figura 7.14, de donde obtenemos $\text{card}_E(M)=\sum_{e_i \in E} \mu_M(e_i)=4$. Si tomamos como definición del cuantificador $Q=“aproximadamente cinco”$ el número borroso definido en la parte izquierda de la Figura 7.13, tenemos que: $g=\mu_Q(4)=1$. Por tanto, el predicado es cierto.

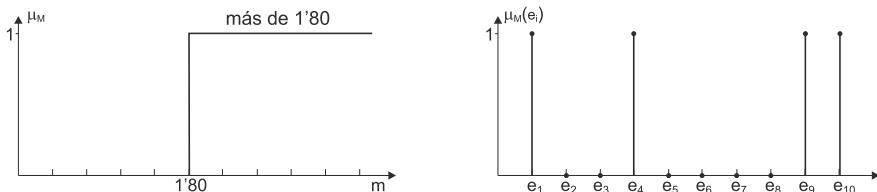


Figura 7.14: Representación gráfica del conjunto preciso que modela el significado del predicado “mide más de 1'80m”, y satisfacción de este predicado para cada uno de los elementos del conjunto de referencia E .

Ejemplo 7.2. Consideremos ahora la evaluación de la sentencia “la gran mayoría del congreso rechazó la moción”, sobre el conjunto de referencia del Congreso de los Diputados $E=\{e_1, \dots, e_{350}\}$, siendo el número de diputados de 350. Supongamos que la votación ha arrojado el resultado de 252 votos en contra de la moción. Eso significa que $\text{card}_E(R)=252$, donde $R=“rechazó la moción”$. Si tomamos como definición del cuantificador $Q=“la gran mayoría”$ el número borroso definido en la parte derecha de la Figura 7.13, tenemos que: $g=\mu_Q(252/350)=\mu_Q(0'72)=0'86$. Por tanto, el predicado es cierto con grado 0'86.

Los dos ejemplos anteriores coinciden en evaluar la satisfacción de un predicado preciso. En el ejemplo siguiente evaluamos un predicado vago, que modelamos mediante un conjunto borroso.

Ejemplo 7.3. Consideremos ahora el predicado “aproximadamente cinco personas son altas”, donde definimos la función de pertenencia del término alto en la Figura 7.15, y donde el conjunto de referencia y los valores de estatura coinciden con los del ejemplo 1. En esa situación, podemos representar gráficamente el cumplimiento del predicado $A = \text{“es alto”}$, para cada elemento del conjunto de referencia, tal y como se indica en la Figura 7.15, de donde obtenemos $\text{card}_E(A) = \sum_{e_i \in E} \mu_A(e_i) = 9$. Si tomamos como definición del cuantificador $Q = \text{“aproximadamente cinco”}$ el número borroso definido en la parte izquierda de la Figura 7.13, tenemos que: $g = \mu_Q(9) = 0$. Por tanto, el predicado es falso.

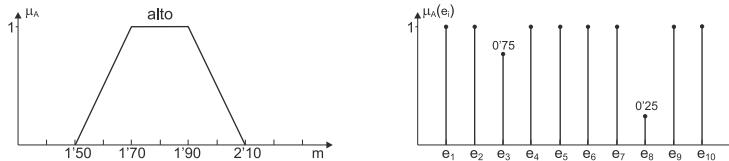


Figura 7.15: Representación gráfica del conjunto borroso que modela el significado del predicado “alto”, y satisfacción de este predicado para cada uno de los elementos del conjunto de referencia E .

Hasta ahora hemos estudiado sentencias unitarias, esto es, aquellas en las que se analiza la satisfacción de un único predicado. Podríamos plantearnos el estudiar sentencias con más de un predicado. Para una sentencia cuantificada binaria del tipo “ Q B son A”, podemos definir la cardinalidad relativa de A en B mediante la siguiente expresión:

$$\text{card}_E(A/B) = \frac{\sum_{e \in E} T(\mu_A(e), \mu_B(e))}{\sum_{e \in E} \mu_B(e)}$$

donde T es una t-norma cualquiera.

Ejemplo 7.4. Consideremos la evaluación de la sentencia “aproximadamente cinco personas jóvenes son altas” sobre el mismo conjunto de referencia de personas del ejemplo 3, donde en esta ocasión conocemos ya directamente el grado de pertenencia de sus edades al conjunto $J = \text{“joven”}$:

$$\mu_J = \{1, 0'9, 1, 0'3, 0, 1, 1, 0'1, 0'6, 1\}$$

Con estos datos, y si tomamos como t-norma el mínimo, tenemos que

$$\min(\mu_A(e), \mu_J(e)) = \{1, 0'9, 0'75, 0'3, 0, 1, 1, 0'1, 0'6, 1\}$$

Por tanto, la cardinalidad relativa de A en J se obtiene mediante el siguiente cálculo:

$$\text{card}_E(A/J) = \frac{\sum_{e \in E} \min(\mu_A(e), \mu_J(e))}{\sum_{e \in E} \mu_J(e)} = \frac{6'65}{6'9} = 0'9638$$

Si nuevamente tomamos como definición del cuantificador Q = “aproximadamente cinco” el número borroso definido en la Figura 7.13 (relativo a un conjunto de referencia de 10 elementos), tenemos que: $g = \mu_Q(0'9638) = 0$. Por tanto, el predicado es falso.

La idea básica de Zadeh de modelar la evaluación de sentencias cuantificadas mediante la compatibilidad entre el cuantificador y la cardinalidad de un conjunto borroso, que representa la satisfacción de un predicado para un conjunto de referencia dado, ha sido seguida por otros autores [Barro y otros, 2003]. La necesidad de disponer de otros modelos aparece desde el momento en que se detecta que el propuesto por Zadeh no cumple algunas propiedades deseables (véanse los ejercicios resueltos 7 y 8), buscando así definir modelos que sí las cumplan. En [Barro y otros, 2003] puede verse una revisión completa de los modelos que utilizan diferentes medidas de cardinalidad, junto con un análisis de sus propiedades. Una vez caracterizadas las propiedades que verifica y las que no verifica cada modelo se dispone de una valiosa herramienta de guía en cuanto a su utilización para la resolución de un problema concreto.

En cualquier caso, la riqueza expresiva de los cuantificadores va mucho más allá de la dicotomía absoluto/relativo recogida en el modelo de Zadeh. Aun cuando la lingüística contempla un buen número de clases de cuantificadores (denominados *determinantes* en ese ámbito), como los *simples* (“alguno”, “estos”), *cardinales* (“entre cinco y diez”, “aproximadamente seis”), *aproximativos* (“casi todos”), *definidos* (“estos diez”), *de excepción* (“todos menos diez”), *proporcionales* (“muchos”, “menos de la mitad”), *negados* (“no todos”), ... que sí encajan en alguno de esos dos tipos, existen otras clases que requieren otros modelos distintos de los sugeridos por Zadeh: cuantificadores *comparativos* (“más trabajadores rubios que jóvenes”), además de los cuantificadores *ternarios* (“todos los estudiantes excepto los de IA están en la fiesta”) y *cuaternarios* (“más estudiantes han venido a la fiesta que profesores han ido a la playa”) o cuantificadores *anidados* (“a pocos niños les gustan todos los helados”).

Hemos presentado en esta sección una aproximación a la cuantificación que pretende modelar, mediante conjuntos borrosos, el significado de aquellas sentencias en las que aparece alguna forma imprecisa de cuantificación. Sin embargo, podemos concebir un uso de los cuantificadores borrosos que hace el camino en el sentido inverso: a partir de un conjunto de datos, podemos utilizar los cuantificadores como un mecanismo que permite realizar un resumen de los datos en sentencias útiles para el usuario humano, en ámbitos como la minería de datos o la extracción de conocimiento, pero también para la agregación de información en la toma de decisiones o para flexibilizar los lenguajes de consultas en sistemas de información (bases de datos, buscadores Web, etc).

7.12 Lecturas recomendadas

Tres buenos libros introductorios a los conjuntos borrosos son [Dubois y Prade, 1980], [Nguyen y Walker, 1996] y [Klir y otros, 1996]. [Buckley y Eslami, 2002] introduce el tema mediante numerosos ejemplos y ejercicios. En [Dubois y Prade, 2000] se realiza una compilación de trabajos de distintos autores que pretenden dar una perspectiva amplia de los fundamentos de los conjuntos borrosos. Dos libros más centrados en los aspectos propios de la lógica son [Trillas y otros, 1995] y [Zadeh y Kacprzyk, 1992]. Los trabajos de Zadeh aparecen compilados en los volúmenes [Yager y otros, 1987] y [Klir y Yuan, 1996].

7.13 Resumen

Desde su primera formalización en 1965, los conjuntos borrosos han pasado a consolidarse como uno de los ejes principales en el desarrollo de la IA. La clave de esta consolidación la encontramos, por un lado, en la rápida aparición de aplicaciones industriales que adoptaron con éxito los conjuntos borrosos, y que dieron por resultado soluciones más flexibles, más eficientes y más económicas. Hoy en día, los conjuntos borrosos están presentes en una amplia variedad de productos de electrónica de consumo, control industrial, sistemas de visión, instrumentación médica, sistemas de información, sistemas de procesado de señal o sistemas de ayuda a la toma de decisión, entre otros. Por otro lado, el esfuerzo realizado en dotar a los conjuntos borrosos de unos fundamentos formales, han contribuido a disponer de un cuerpo de conocimiento cuyos principios son suficientemente sólidos como para dar lugar a soluciones confiables.

La principal motivación que nos lleva a los conjuntos borrosos se encuentra en la dificultad que entraña resolver una gran parte de los problemas del mundo real mediante las herramientas tradicionales de la ciencia y la ingeniería. Problemas como conducir, clasificar, ordenar o planificar, se realizan habitualmente en unas condiciones que impiden proporcionar una solución analítica. Estas condiciones afectan fundamentalmente al acceso a la información del entorno, cuya adquisición se ve aquejada de imprecisión e incertidumbre. Sin embargo, los seres humanos realizamos estas tareas con una aparente facilidad. Los conjuntos borrosos suponen un cambio de perspectiva, por el que pasamos de modelar un problema, objeto tradicional de la ciencia, a modelar la forma en que un ser humano lo resuelve, objeto de una nueva disciplina que recibe el nombre de ingeniería del conocimiento.

En este capítulo hemos querido poner énfasis en que casi todo lo que hacemos con los conjuntos borrosos responde a una cuestión de diseño, de modo que podemos concebir distintas alternativas basadas en conjuntos borrosos para modelar el conocimiento y las herramientas que permiten su manipulación. Este diseño está sujeto a algunas propiedades que exigimos para asegurar la consistencia en las operaciones que involucran a este conocimiento, y que nos deben impedir obtener resultados contrarios a la lógica. Sin embargo, esta lógica ya no es la lógica clásica, sino un nuevo instrumento que damos en llamar lógica borrosa, que construimos sobre una

teoría no estándar de conjuntos, y que ha de proporcionar principios normativos al razonamiento aproximado.

Desde la perspectiva del diseño y desarrollo de sistemas basados en conocimiento, los conjuntos borrosos forman parte de un conjunto más amplio de herramientas que pertenecen al ámbito de lo que se conoce como *Computación Flexible (Soft Computing)*, y cuyo objetivo es el desarrollo de soluciones que proporcionen a la computación una mayor tolerancia a la imprecisión, la incertidumbre y la verdad parcial. Entre estas herramientas encontramos las Redes Neuronales Artificiales, los Algoritmos Evolutivos, y el Razonamiento Probabilístico.

Lejos de ser excluyentes entre sí, las distintas herramientas de la Computación Flexible pueden participar juntas formando parte de soluciones híbridas, que aprovechan las ventajas de complementar sus cualidades más relevantes. Pongamos por caso el problema de la clasificación, introducido en la sección 7.10.1. Los conjuntos borrosos nos permiten diseñar un clasificador a partir de una especificación lingüística, basada probablemente en un conocimiento que proviene de la experiencia. Sin embargo, no siempre resulta sencillo proporcionar un conocimiento en términos de una relación exhaustiva entre antecedentes y consecuentes. Se nos ocurre que podemos incorporar algoritmos de aprendizaje inductivo, como el de retropropagación, que dan lugar a las llamadas *reglas neuroborrosas*, en las que el aprendizaje tiene como objetivo el determinar la forma de las funciones de pertenencia y las reglas del clasificador [Kwan y Cai, 1994]. Otro formalismo que permite la construcción automática de sistemas de reglas borrosas para problemas de clasificación es el de los *algoritmos evolutivos*. Constituyen algoritmos de búsqueda global de soluciones mediante una población de individuos. La búsqueda se identifica con la evolución de la población, evolución que tiene lugar mediante procesos de mutación y cruce. En el caso particular de la clasificación, un individuo puede ser un conjunto de reglas condicionales borrosas, y la evolución de la población se puede interpretar como un proceso de optimización que ha de obtener un clasificador con una alta capacidad de generalización y un número reducido de reglas (véase capítulo 18).

En último término, la principal ventaja de una representación en forma de base de reglas condicionales borrosas es su capacidad para proporcionar un resultado en el que el propio mecanismo de obtención resulta fácilmente comprensible para un usuario humano.

7.14 Ejercicios resueltos

7.1. Demostrar que el operador máximo es la t-conorma dual del operador mínimo.

Solución: En efecto. Hemos definido la t-conorma dual de una t-norma mediante la expresión: $S(x, y) = n(T(n(x), n(y)))$, donde $n(x) = 1 - x$. Cuando $T(x, y) = \min(x, y)$ tenemos:

$$S(x, y) = 1 - \min(1 - x, 1 - y)$$

Si $x \geq y$ entonces, $1 - x \leq 1 - y$, luego $\min(1 - x, 1 - y) = 1 - x$ y, por tanto, $S(x, y) = 1 - (1 - x) = x$. Si $x \leq y$, entonces $S(x, y) = y$. Por consiguiente, $S(x, y) = \max(x, y)$, tal y como queríamos demostrar.

■

7.2. Representar mediante conjuntos borrosos el significado de la sentencia “menos de aproximadamente 15 voltios mayor”.

Solución: Una forma muy común de representar un conjunto borroso C es mediante una distribución trapezoidal $C = (\alpha, \beta, \gamma, \delta)$, $\alpha \leq \beta \leq \gamma \leq \delta$, donde el intervalo $[\beta, \gamma]$ se corresponde con el núcleo de C y el intervalo $]\alpha, \delta[$ se corresponde con su soporte.

Supongamos ahora que el dominio de valores de la tensión es discreto. Es una suposición realista, ya que hoy en día la gran mayoría de los instrumentos es digital. Exageremos un poco esta suposición, haciendo que los valores de la tensión sean números enteros.

Podemos representar así el valor “15 voltios” mediante el conjunto borroso $V = (15, 15, 15, 15)$. Si representamos el término “aproximadamente” mediante el conjunto $A = (-3, -1, 1, 3)$, el significado de la sentencia “aproximadamente 15 voltios” viene dado por la suma borrosa $A \oplus V = (-3, -1, 1, 3) \oplus (15, 15, 15, 15) = (12, 14, 16, 18)$.

La sentencia “menos de aproximadamente 15 voltios” se puede modelar mediante la resta borrosa $(12, 14, 16, 18) \ominus (1, 1, \infty, \infty) = (-\infty, -\infty, 15, 17)$. Y, por último, la sentencia final “menos de aproximadamente 15 voltios mayor” puede utilizar como significado de “mayor” el conjunto borroso $(1, 1, \infty, \infty)$, de modo que el significado de la sentencia completa se calcula como la intersección $(-\infty, -\infty, 15, 17) \cap (1, 1, \infty, \infty) = (1, 1, 15, 17)$

■

7.3. ¿Cómo se comporta la relación de diferencia dada por la expresión $\mu_D(u, v) = 1 - \mu_S(u, v)$, donde S es una relación de similitud?

Solución: Supongamos que la relación S satisface la propiedad de transitividad para la t-norma mínimo. En ese caso tenemos:

$$\min(\mu_S(u, v), \mu_S(v, w)) \leq \mu_S(u, w), \quad \forall u, v, w \in U$$

Al ser el máximo la t-conorma dual del mínimo podemos escribir de modo equivalente:

$$1 - \max(1 - \mu_S(u, v), 1 - \mu_S(v, w)) \leq \mu_S(u, w)$$

De la definición de $\mu_D(u, v) = 1 - \mu_S(u, v)$ tenemos:

$$\max(\mu_D(u, v), \mu_D(v, w)) \geq \mu_D(u, w)$$

Si se satisface esta desigualdad entonces también se satisface la desigualdad triangular:

$$\mu_D(u, v) + \mu_D(v, w) \geq \mu_D(u, w)$$

Tenemos, por tanto, que una relación de similitud que satisface la t-transitividad para el mínimo permite generar una relación de distancia más restrictiva que una relación métrica, denominada ultramétrica [Dubois y Prade, 1980]. Si la t-norma de la propiedad de t-transitividad es la de Lukasiewicz se puede demostrar fácilmente que obtenemos una métrica clásica restringida al intervalo $[0, 1]$.

■

7.4. ¿Existe alguna relación entre las relaciones de implicación e inclusión?

Solución: En efecto. En el caso preciso es bien sabido que $A \subseteq B \Leftrightarrow A \rightarrow B$, o dicho de otro modo, decir que A es subconjunto de B es equivalente a decir que “si x es A , entonces x es B ”. Parece razonable pensar que esto mismo es válido cuando los predicados son vagos, como es el caso en que A =“muy rojo” y B =“rojo” (véase la Figura 7.3), por lo que podemos pensar en el uso de las implicaciones como relaciones de inclusión entre conjuntos borrosos.

■

7.5. Probar que la ley de contraposición $(A \rightarrow B) \Leftrightarrow (\neg B \rightarrow \neg A)$, no es cierta para la implicación de Gödel, esto es, que dicha implicación no satisface la propiedad I11.

Solución: En efecto. Si tomamos la negación $n(x) = 1 - x$ tenemos:

$$\mu_{I(a,b)} = \begin{cases} 1 & \text{si } a \leq b \\ b & \text{si } a > b \end{cases} \quad \mu_{I(n(b),n(a))} = \begin{cases} 1 & \text{si } a \leq b \\ 1 - a & \text{si } a > b \end{cases}$$

Luego $I(a, b) \neq I(n(b), n(a))$.

■

7.6. ¿Qué podemos concluir a partir de la regla “Si el metal está rojo entonces el radiador está caliente” cuando se observa que “el metal está muy rojo”? Suponer la regla composicional de inferencia formada por la implicación de Kleene-Dienes y la t-norma de Lukasiewicz.

Solución: Sea A =ROJO, B =CALENTE y A' =MUY ROJO. Supongamos que $A' \subseteq A$. La Tabla 7.3 nos dice que la regla composicional propuesta en el enunciado satisface la propiedad MP7, luego $B' = B$, esto es, la conclusión sigue siendo “el radiador está caliente”. Para demostrar este resultado supongamos que A' está normalizado. La regla composicional de inferencia tiene en este caso la siguiente expresión:

$$\begin{aligned}
 \mu_{B'}(v) &= \sup_{u \in U} \max(\mu_{A'}(u) + \max(1 - \mu_A(u), \mu_B(v)) - 1, 0) \\
 &= \sup_{u \in U} \max(\max(\mu_{A'}(u) - \mu_A(u), \mu_{A'}(u) + \mu_B(v)) - 1, 0) \\
 &= \sup_{u \in U} \max(\mu_{A'}(u) + \mu_B(v) - 1, 0) \\
 &= \max(\sup_{u \in U} (\mu_{A'}(u) + \mu_B(v) - 1), 0) \\
 &= \max(\mu_B(v), 0) \\
 &= \mu_B(v)
 \end{aligned}$$

Tal y como queríamos demostrar.

■

7.7. Probar si el modelo de cuantificación de Zadeh cumple las propiedades de:

- *Negación interna* (antonimia), por la cual resultan equivalentes las expresiones “Q B son A” y “(ant Q) B son no A”. Ejemplo: “Algunos estudiantes jóvenes son altos” \Leftrightarrow “Muchos estudiantes jóvenes son no altos”.
- *Negación externa*, por la cual resultan equivalentes las expresiones “Q B son A” y “no((no Q) B son A)”. Ejemplo: “Menos de tres estudiantes jóvenes son altos” \Leftrightarrow “Es falso que al menos tres estudiantes jóvenes son altos”.

Solución: Consideremos en primer lugar la propiedad de negación interna. Para analizar su cumplimiento, consideremos el predicado $P_2 = “(ant Q)son no A”$, con los operadores de antonimia y negación definidos en la secciones 7.4 y 7.5. En ese caso, tenemos:

$$\begin{aligned}
 g_2 = \mu_{(ant)Q}(card_E(\overline{A}/B)) &= \mu_{(ant)Q}\left(\frac{\sum_{e \in E} T(1 - \mu_A(e), \mu_B(e))}{\sum_{e \in E} \mu_B(e)}\right) \\
 &= \mu_Q\left(1 - \frac{\sum_{e \in E} T(1 - \mu_A(e), \mu_B(e))}{\sum_{e \in E} \mu_B(e)}\right) \\
 &= \mu_Q\left(\frac{\sum_{e \in E} \mu_B(e) - T(1 - \mu_A(e), \mu_B(e))}{\sum_{e \in E} \mu_B(e)}\right)
 \end{aligned}$$

La verificación o no de la propiedad dependerá de la t-norma T que se escoja. Bastará ver si $\mu_B(e) - T(1 - \mu_A(e), \mu_B(e)) = T(\mu_A(e), \mu_B(e))$ para que la propiedad se cumpla. Así, para $T=\min$, tenemos que debería ser: $\mu_B(e) - \min(1 - \mu_A(e), \mu_B(e)) = \min(\mu_A(e), \mu_B(e))$ o, equivalentemente, $\max(\mu_A(e) + \mu_B(e) - 1, 0) = \min(\mu_A(e), \mu_B(e))$. Es fácil encontrar un contraejemplo para ver que dicha igualdad no se verifica. Bastaría tomar $A=\{0.8/e_1\}$ y $B=\{0.2/e_1\}$. Sin embargo, para $T=\text{producto}$, tenemos: $\mu_B(e) - (1 - \mu_A(e)) \cdot \mu_B(e) = \mu_A(e) \cdot \mu_B(e)$, por lo que la propiedad se cumple. Es fácil ver que la propiedad tampoco se cumple para la t-norma de Lukasiewicz.

Consideremos ahora la propiedad de negación externa. Para analizar su verificación analicemos el predicado $P_2=\text{"no (no Q) B son A"}$, con el operador de negación definido en la sección 7.4. En este caso tenemos:

$$g_2 = 1 - (\mu_{(\text{no})Q}(\text{card}_E(A/B))) = 1 - (1 - \mu_Q(\text{card}_E(A/B))) = \mu_Q(\text{card}_E(A/B))$$

Por tanto, la propiedad se cumple independiente de cuál sea el operador t-norma que se utilice. ■

7.8. Una de las críticas principales al modelo de Zadeh es su efecto acumulativo, por el que el efecto de varios elementos con grado de pertenencia bajo es igual al de uno con grado de pertenencia alto. Valorar dicho efecto evaluando la sentencia “existe una persona alta” para un conjunto de referencia como el de la Figura 7.16.

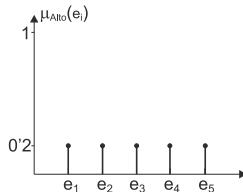


Figura 7.16: Representación gráfica de la satisfacción del predicado “persona alta” para cada uno de los elementos de un conjunto de referencia de cinco individuos.

Solución: Para el predicado considerado tenemos que

$$g = \mu_{\exists}(\sum_{e \in E} \mu_{ALTO}(e)) = \mu_{\exists}(1) = 1$$

Por tanto, el predicado es cierto. Sin embargo, a pesar de lo contraintuitivo que pueda parecer este resultado, debemos tener en cuenta que el cuantificador “existe” tiene una semántica de t-conorma (algo evidente si pensamos en el caso preciso). Si escogiésemos la t-conorma de Lukasiewicz para modelar este mismo ejemplo obtendríamos el mismo resultado. ■

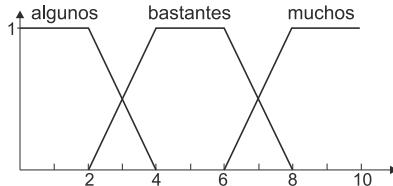


Figura 7.17: Representación gráfica de los cuantificadores del conjunto C.

7.15 Ejercicios propuestos

- 7.1.** Demostrar que todas las t-normas se comportan igual en el caso preciso, esto es, cuando sus argumentos toman valores en $\{0, 1\}$.
- 7.2.** Demostrar que el producto y la suma probabilística son respectivamente duales.
- 7.3.** Demostrar que cualquier pareja formada por una t-norma y su t-conorma dual satisfacen las leyes de De Morgan.
- 7.4.** Probar las igualdades $(A \cap B)_\alpha = A_\alpha \cap B_\alpha$ y $(A \cup B)_\alpha = A_\alpha \cup B_\alpha$ con las distintas t-normas y t-conormas introducidas en el capítulo.
- 7.5.** Calcular el grado de compatibilidad entre los conjuntos borrosos “muy rojo” y “rojo”.
- 7.6.** Representar mediante conjuntos borrosos el significado de la sentencia “poco menos de aproximadamente 15 minutos antes”.
- 7.7.** Comprobar que podemos utilizar la implicación como índice de inclusión para las distintas implicaciones de la Tabla 7.2.
- 7.8.** Demostrar que la cardinalidad de Zadeh para los cuantificadores absolutos es coherente con el caso preciso.
- 7.9.** A partir del conjunto de cuantificadores $C = \{\text{“algunos”}, \text{“bastantes”}, \text{“muchos”}\}$, mostrados en la Figura 7.17, podemos obtener nuevos cuantificadores mediante alguna combinación que involucre operaciones de conjunción, disyunción, negación o antónimo. Se pide obtener de este modo los cuantificadores “muchos pero no bastantes”, “no muchos”, y el antónimo de “algunos”.

Referencias

- BARRO, S.; BUGARÍN, A.; CARIÑENA, P. y F.DÍAZ-HERMIDA: «A framework for fuzzy quantification models analysis». *IEEE Transactions on Fuzzy Systems*, 2003, **11**, pp. 89–99.
- BELLMAN, R.E. y ZADEH, L.A.: «Local and fuzzy logics». En: M.M. Gupta; A. Kandell; W. Bandler y J.B. Kiszka (Eds.), *Modern Uses of Multiple-Valued Logic*, Reidel, 1977.
- BLACK, M.: «Vagueness: an exercise in logical analysis». *Philosophy of Science*, 1937, **4**, pp. 427–455.
- BUCKLEY, J.J. y ESLAMI, E.: *An introduction to fuzzy logic and fuzzy sets*. Physica Verlag, 2002.
- CASTRO, J.L. y DELGADO, M.: «Fuzzy systems with defuzzifications are universal approximators». *IEEE Transactions on System, Man and Cybernetics*, 1996, **26(1)**, pp. 149–152.
- COPÍ, I.: *Introducción a la lógica*. Eudeba, 1999.
- DUBOIS, D. y PRADE, H.: *Fuzzy Sets and Systems: Theory and applications*. Academic Press, 1980.
- DUBOIS, D. y PRADE, H.: «Processing fuzzy temporal knowledge». *IEEE Transactions on Systems, Man and Cybernetics*, 1989, **19(4)**, pp. 729–744.
- DUBOIS, D. y PRADE, H.: «Fuzzy sets in approximate reasoning, Part 1: Inference with possibility distributions». *Fuzzy Sets and Systems*, 1991, **40**, pp. 143–202.
- DUBOIS, D. y PRADE, H. (Eds.): *Fundamentals of fuzzy sets*. Kluwer Academic Publishers, 2000.
- FUKAMI, S.; MIZUMOTO, M. y TANAKA, K.: «Some considerations on fuzzy conditional inference». *Fuzzy Sets and Systems*, 1980, **4**, pp. 243–273.
- GOGUEN, J.A.: «The logic of inexact concepts». *Synthese*, 1969, **19**, pp. 325–373.
- KLIR, G.; CLAIR, U. y YUAN, B. (Eds.): *Fuzzy set theory: Foundations and Applications*. Prentice Hall, 1996.
- KLIR, G. y YUAN, B. (Eds.): *Fuzzy sets, fuzzy logic, and fuzzy systems. Selected papers by Lotfi A. Zadeh*. World Scientific, 1996.
- KUNCHEVA, L.I.: «How good are fuzzy if-then classifiers?» *IEEE Transactions on Systems, Man and Cybernetics*, 2000, **30**, pp. 501–509.
- KWAN, H.K. y CAI, Y.: «A fuzzy neural network and its application to pattern recognition». *IEEE Transactions on Fuzzy Systems*, 1994, **2(3)**, pp. 185–193.

- LAKOFF, G.: «Hedges: a study in meaning-criteria and the logic of fuzzy concepts». *Journal of Philosophical Logic*, 1973, **2**, pp. 458–508.
- LUKASIEWICZ, J.: *Selected works*. Reidel, 1970.
- MIZUMOTO, M. y ZIMMERMANN, H.: «Comparison of fuzzy reasoning methods». *Fuzzy Sets and Systems*, 1982, **8**, pp. 253–283.
- NGUYEN, H.T. y WALKER, E.A.: *A first course in fuzzy logic*. CRC Press, 1996.
- RUSPINI, E.H.: «A new approach to clustering». *Information and Control*, 1969, **15**, pp. 22–32.
- TRILLAS, E.; ALSINA, C. y TERRICABRAS, J.M.: *Introducción a la lógica borrosa*. Ariel Matemática, 1995.
- TRILLAS, E. y VALVERDE, L.: «On implication and indistinguishability in the setting of fuzzy logic». En: J. Kacprzyk y R. Yager (Eds.), *Management decision support systems using fuzzy sets and possibility theory*, Verlag TÜV Rheinland, 1985.
- YAGER, R.R.; OVCHINNIKOV, S.; TONG, R.M. y NGUYEN, H.T. (Eds.): *Fuzzy sets and applications. Selected papers by Lotfi A. Zadeh*. John Wiley & Sons, 1987.
- ZADEH, L.A.: «Fuzzy sets». *Information Control*, 1965, **8**, pp. 338–353.
- ZADEH, L.A.: «Outline of a new approach to the analysis of complex systems and decision processes». *IEEE Transactions on Systems, Man and Cybernetics*, 1973, **3(1)**, pp. 28–44.
- ZADEH, L.A.: «The concept of a linguistic variable and its application to approximate reasoning». *Information Sciences*, 1975, **8**, pp. 199–249.
- ZADEH, L.A.: «Fuzzy sets as a basis for a theory of possibility». *Fuzzy Sets and Systems*, 1978, **1**, pp. 3–28.
- ZADEH, L.A.: «A computational approach to fuzzy quantifiers in natural languages». *Computing and Mathematics with applications*, 1983, **9(1)**, pp. 149–184.
- ZADEH, L.A. y KACPRZYK, J. (Eds.): *Fuzzy logic for the management of uncertainty*. John Wiley & Sons, 1992.

Parte III

TÉCNICAS

Capítulo 8

Introducción a las técnicas de búsqueda

María Camino Rodríguez Vela y Ramiro Varela Arias

Universidad de Oviedo

8.1 Introducción

En IA los términos resolución de problemas y búsqueda se refieren a un núcleo fundamental de técnicas que se utilizan en dominios como la deducción, elaboración de planes de actuación, razonamientos de sentido común, prueba automática de teoremas, etc. Aplicaciones de estas ideas generales aparecen en la práctica totalidad de los sistemas inteligentes, como por ejemplo en los programas que tratan de entender el lenguaje natural, en los programas que tratan de sintetizar un conjunto de reglas de clasificación en un determinado dominio de actuación, o en los sistemas que realizan inferencias a partir de un conjunto de reglas. En este capítulo y en el siguiente se examinarán los algoritmos de búsqueda como una herramienta para resolver problemas y se tratarán de un modo general con objeto de que sirvan de base para abordar una variedad de problemas de distinta naturaleza. En el capítulo 10 se verán técnicas de búsqueda específicas para un tipo particular de problemas, como son los problemas de satisfacción de restricciones. Finalmente, en el capítulo 11, se verán también técnicas de resolución de problemas basadas en computación evolutiva.

La resolución de problemas en IA requiere, normalmente, determinar una secuencia de acciones o decisiones. Esta secuencia será ejecutada posteriormente por un agente con el fin de alcanzar un objetivo a partir de una situación inicial dada. Dependiendo del problema concreto, la ejecución de la secuencia de acciones o decisiones tiene asociado un coste que se tratará de minimizar, o bien tiene asociado un beneficio que se tratará de maximizar. En la descripción de los sistemas de búsqueda que se realiza en este capítulo, se supondrá que el agente se mueve en un entorno accesible, o lo que es lo mismo, que es capaz de percibir el entorno con precisión y que tanto el efecto como el coste (o el beneficio) de las acciones se pueden predecir con exactitud. De este modo, la secuencia de acciones se puede obtener antes de su ejecución; en otro

caso, la siguiente acción no podría ser determinada hasta conocer el resultado de la ejecución de la anterior.

8.2 Algunos ejemplos

Con el fin de introducir las ideas y componentes esenciales de los sistemas de búsqueda, en esta sección se introducen algunos ejemplos clásicos que se resuelven con este tipo de algoritmos. En general se trata de problemas NP-duros, es decir problemas para los que no se conoce un algoritmo polinomial que los resuelva y que, además, dada una solución, no es posible verificar en tiempo polinomial que se trata de una solución óptima. El objetivo es mostrar distintos tipos de problemas de búsqueda y caracterizar los métodos de aplicación más general.

8.2.1 Generación de planes de actuación de robots

Este ejemplo está tomado de [Nilsson, 2001], sección 7.2. La investigación sobre resolución de problemas de robots ha dado lugar a muchas técnicas de resolución de problemas generales que se usan en IA. El planteamiento típico de uno de estos problemas consta de un robot que tiene un repertorio finito de acciones que puede ejecutar y de un entorno de actuación. La situación más paradigmática es la del “mundo de bloques”. En este caso se dispone de unos cuantos bloques distinguibles unos de otros situados sobre una mesa y un brazo móvil (el robot) capaz de cambiarlos de sitio; se supone, además, que el robot es capaz de localizar a estos bloques al objeto de realizar alguna tarea con ellos.

Programar un robot consiste en integrar varias funciones, entre las que se incluye la percepción del entorno que le rodea, la formulación de planes de actuación y el seguimiento de esos planes. Para ello el primer paso es disponer de un modelo del entorno que permita representar lo relevante de todas las situaciones que se puedan producir.

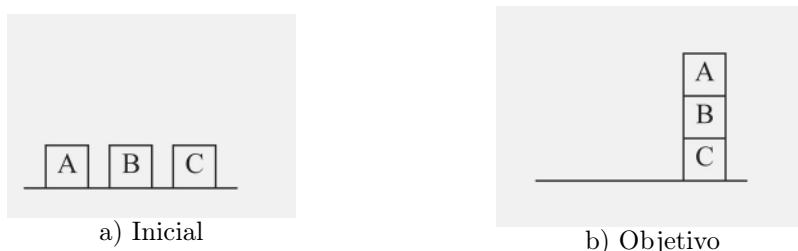


Figura 8.1: Estados inicial y objetivo para una instancia del problema de generación de planes de actuación de robots con tres bloques.

Supóngase que el entorno está formado por tres bloques A, B y C, inicialmente situados en el suelo, y que el objetivo es llegar a una situación en la que los tres bloques estén formando una pila de modo que el bloque C esté en el suelo, el bloque

B sobre C y el bloque A sobre B, tal y como se muestra en la Figura 8.1. El modelo del entorno debe permitir representar lo relevante de éstas, y otras, situaciones. En este caso lo relevante es el hecho de que un bloque esté sobre otro o sobre el suelo y que un bloque tenga a no otro bloque encima; sin embargo, la posición concreta del suelo en la que está un bloque no se considera relevante. Esto es así porque se supone que el sistema de percepción del robot le permite identificar distintas configuraciones de los bloques con independencia del lugar que ocupen respecto al suelo. De este modo, un modelo basado en listas permitirá representar los estados anteriores como $((A)(B)(C))$ y $((A\ B\ C))$ respectivamente. Se supone también que el robot es capaz de mover cualquier bloque, x , que no tiene otro bloque sobre él, a otra posición, el suelo o bien otro bloque y que tampoco tiene un bloque sobre él.

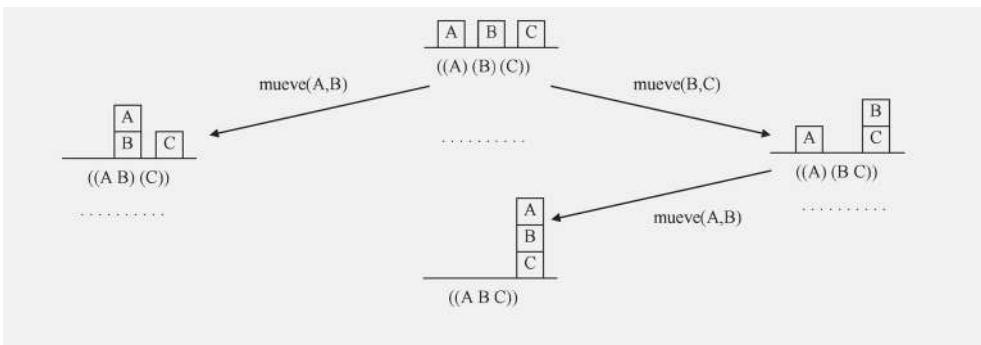


Figura 8.2: Ejemplos de aplicación de algunos operadores a partir del estado *inicial* para un problema de planificación de actuaciones.

Las acciones elementales se pueden representar mediante instancias del esquema $\text{mueve}(x,y)$, donde x representa uno de los bloques e y representa un bloque o bien el suelo. Evidentemente no todas las instancias de este esquema se pueden aplicar a cualquier estado, e incluso algunas no tienen sentido alguno, como por ejemplo $\text{mueve}(A,A)$. Por lo tanto, será necesario dar una especificación precisa del efecto de las instancias permitidas de este esquema mediante una descripción genérica del estado en el que se aplican y, en cada caso, del estado resultante de la aplicación del operador. La Figura 8.2 muestra el efecto de algunos operadores que se pueden aplicar a partir del estado *inicial* de la Figura 8.1 (a). Como se puede observar, en este caso hay un total de 6 operadores aplicables al estado *inicial* y, en cualquier caso, los operadores son reversibles, por ejemplo si al estado resultante de aplicar el operador $\text{mueve}(A,B)$ al estado *inicial* se le aplica el operador $\text{mueve}(A,\text{suelo})$ se obtiene de nuevo el estado *inicial*. La solución para esta instancia del problema es muy simple, se trata de la secuencia $(\text{mueve}(B,C), \text{mueve}(A,B))$. Es muy fácil identificar de forma visual esta solución en el gráfico de la Figura 8.2, sin embargo un programa de ordenador debe considerar las 6 posibilidades resultantes del estado *inicial* y las que se puedan generar a partir de éstas de una forma sistemática hasta llegar a una situación objetivo. Obviamente, para instancias de mayor tamaño, es decir con un número mayor de bloques, la cantidad de situaciones intermedias crece

de forma exponencial con lo que el cálculo de la secuencia de acciones que lleva a la solución es un problema realmente complicado, particularmente si el objetivo es una secuencia óptima, es decir con el menor número de acciones posible.

Este problema permite entender cómo son dos de los elementos esenciales de un sistema de búsqueda: la representación de los estados y la representación de los operadores o reglas de producción. Los estados modelan las situaciones del entorno de actuación del agente, y las reglas modelan las acciones elementales del agente sobre el entorno. Generalmente, las reglas tienen asociado un coste, o, en ocasiones, un beneficio. En este caso, lo natural es interpretar que cada regla tiene asociado un coste debido al movimiento de un bloque. La interpretación concreta del coste depende de la función objetivo que se trata de minimizar. Dado que en este caso se trata de minimizar el número de movimientos, lo natural es asignar a todas las reglas un coste igual a 1.

La descripción precisa de los estados y de las reglas, con sus costes, define el espacio de búsqueda. En el ejemplo del robot en un mundo de bloques, este espacio tiene forma de grafo dirigido simple. En particular, este grafo tiene ciclos y dado un par de nodos puede haber varios caminos entre ellos con distinta longitud. Una vez que se tiene definido el espacio de búsqueda, el paso siguiente es definir la tercera de las componentes principales de un sistema de búsqueda: la estrategia de control o algoritmo de búsqueda. La estrategia de control indica el orden en el que se van a visitar los nodos a partir del nodo inicial para llegar a un nodo objetivo. Como es natural la estrategia de control es el elemento más importante desde el punto de vista de la IA. Una buena estrategia de control debe permitir llegar a una solución sin necesidad de visitar todos, o casi todos, los nodos del espacio de búsqueda. Para ello, como se verá en el capítulo siguiente, será preciso utilizar conocimiento sobre el dominio del problema. En este capítulo se estudian solamente algunas estrategias que no utilizan este tipo de información, por lo que se denominan también estrategias de búsqueda a ciegas. En cualquier caso, las estrategias de búsqueda a ciegas son importantes para establecer comparaciones con las estrategias de búsqueda informada o búsqueda inteligente.

8.2.2 Problemas de rutas óptimas en grafos

La teoría de grafos presenta algunos ejemplos típicos de búsquedas. El más sencillo consiste en encontrar el camino más corto entre dos nodos dados suponiendo que cada arco, o cada eje, tiene asociado un coste o longitud. Se trata, seguramente, del problema más fácil de formular, ya que el propio grafo define el espacio de búsqueda: los nodos representan los estados o situaciones por las que podemos pasar para ir de un nodo inicial a otro final u objetivo, y los arcos representan las transiciones con sus costes. El problema es, por supuesto, encontrar una estrategia de control adecuada. El objetivo de una buena estrategia de control es conseguir una solución, óptima si es posible y si no semióptima, sin necesidad de visitar un número muy grande de nodos.

Otro problema famoso de búsqueda en grafos es el problema del viajante de comercio conocido también como el TSP (Traveling Salesman Problem). En este problema, dado un grafo que representa a un conjunto de ciudades y sus conexiones, se trata de

buscar la ruta que debe seguir un viajante para visitarlas todas y volver a la ciudad de partida con el menor coste posible. En este caso, no hay que confundir el grafo que representa los datos del problema, con el grafo que representa el espacio de búsqueda. Los nodos de este espacio vendrán definidos por secuencias de ciudades visitadas en un momento dado y las reglas representarán enlaces con las ciudades a ser visitadas posteriormente. En el ejercicio resuelto 2 de la sección 8.6 se describe una formulación precisa de este problema en el marco de la búsqueda en espacios de estados.

8.2.3 Juegos con contrincante

En los juegos de dos contrincantes, como las damas, el ajedrez, el tres en raya, etc., cada jugador realiza una búsqueda entre todas las jugadas que puede hacer en cada instante de tal forma que la situación resultante de la jugada le sea lo más favorable posible. En este caso las reglas del juego marcan las posibilidades de acción y el objetivo es conceptualmente sencillo: alcanzar una situación de victoria. En este tipo de búsquedas hay, en principio, una diferencia sustancial con respecto a los problemas que se han considerado en las secciones anteriores: el agente sólo tiene el control de forma alterna ya que el contrincante también juega. Con este planteamiento, existen sistemas de búsqueda específicos para estos problemas basados en algoritmos como el método *MinMax* y el procedimiento de poda $\alpha - \beta$ [Pearl, 1984]. Con estos algoritmos, el espacio de búsqueda tiene el aspecto que se muestra en la Figura 8.3. Se trata de un árbol en el que los dos jugadores, A y B, tienen el control de forma alterna.

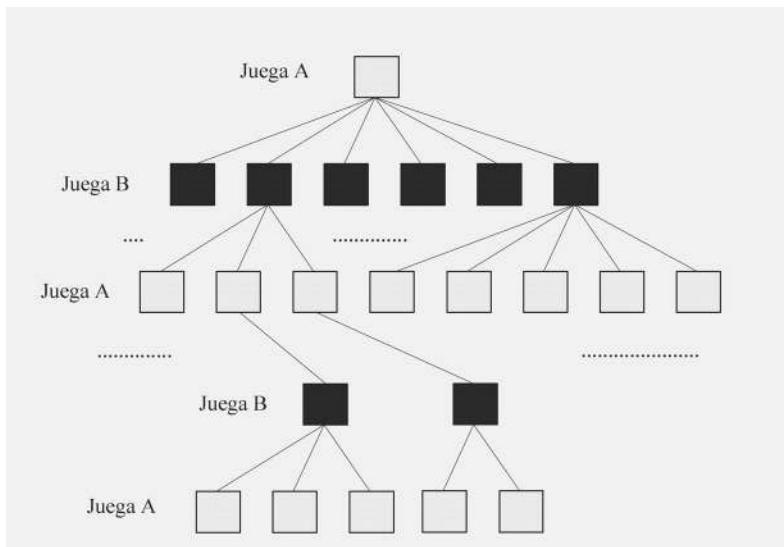


Figura 8.3: Espacio de búsqueda para problemas de juegos con movimientos alternos.

Sin embargo, en este caso el objetivo de la búsqueda no puede ser un camino simple desde el estado *inicial* a un estado objetivo de victoria para el jugador A.

En principio se puede tratar de encontrar lo que se denomina una *estrategia ganadora* que viene definida por un subárbol tal que para cada nodo en el que mueve A incluye un único sucesor en el que mueve B, para cada nodo en el que mueve B incluye todos sus sucesores, y así hasta llegar a nodos que representen situaciones de victoria para A, o al menos tablas. Como es de esperar, en juegos como las damas o el ajedrez, el tamaño del espacio de búsqueda, y también de una estrategia ganadora, viene dado por cifras astronómicas (del orden 10^{120}) por lo que una búsqueda exacta es inabordable. Lo único que es posible en la práctica es analizar una parte pequeña del espacio de búsqueda, a partir de la situación actual, para predecir el movimiento más adecuado y luego esperar al movimiento del contrincante para predecir la siguiente jugada. Aunque se conocen perfectamente todos los estados posibles y el resultado de todas las acciones, debido a que el número de estados es extremadamente grande, hay que proceder como si el resultado de las acciones fuese impredecible, de modo que se debe intercalar la búsqueda con la ejecución real de las acciones.

8.3 Formulación de problemas de búsqueda

Este capítulo y el siguiente se centran fundamentalmente en los métodos de búsqueda en espacios de estados, ya que en la práctica este paradigma es el de aplicación más general. En esta sección se resumen los elementos más importantes que deben ser considerados en el diseño de un sistema de búsqueda en espacios de estados para resolver un determinado problema. Como ya se ha adelantado en las secciones anteriores las componentes principales de un sistema de búsqueda son tres: los estados, las reglas u operadores y la estrategia de control.

El conjunto de estados representa todas las situaciones por las que el agente puede eventualmente pasar durante la solución del problema, como por ejemplo las configuraciones de los bloques en el problema de planificación de robots, o las secuencias parciales de ciudades en el TSP. En consecuencia, hay que comenzar por elegir un modelo de representación con un nivel de detalle adecuado a las capacidades de percepción y actuación del agente. Los operadores deben modelar las acciones elementales que es capaz de realizar el agente y dependen obviamente del modelo elegido para los estados. Cada operador debe quedar perfectamente definido a partir de una especificación precisa de las condiciones que debe cumplir el estado en el que se aplica y del estado resultante de su aplicación. La interpretación del coste de cada operador debe hacerse a partir de la función objetivo; en el caso de problemas de minimización solamente es preciso responder a la pregunta ¿cuánto aumenta la cantidad que se quiere minimizar en la solución al pasar de un estado a otro estado como consecuencia de aplicar el operador? En el problema del mundo de bloques el coste de todas las reglas es 1, pero en el problema del TSP debe ser el coste de ir de la última ciudad de la secuencia a la que se visita a continuación.

El conjunto de estados y operadores definen el espacio de búsqueda que, en el paradigma de búsqueda en espacios de estados, tiene forma de grafo dirigido simple. En los problemas de interés real, este grafo es de un tamaño tan grande que no es posible representarlo de forma explícita en la memoria del ordenador, sino que queda

representado de forma implícita a partir de un único estado inicial y del conjunto de operadores. En general, hay varios nodos del espacio de búsqueda que representan soluciones del problema; estos son los objetivos. En la práctica tampoco es posible almacenarlos de forma explícita ya que en general no se conocen o son muchos, por lo que es preciso disponer de una función que permita caracterizarlos. Por ejemplo, en el TSP, aunque cualquier permutación de las ciudades es una solución del problema, supuesto que el grafo de conexiones es totalmente conexo, no es adecuado almacenar todas estas soluciones en memoria, sino que lo que procede es tener una función que compruebe si el estado contiene a todas las ciudades, para determinar que se trata de un objetivo.

La estrategia de control es la responsable de decidir el orden en el que se van explorando los estados, o lo que es igual, el orden en el que se va considerando la aplicación de los operadores. Una estrategia de control inteligente, o bien informada, debería llevar, con una alta probabilidad, a explorar primero aquellos nodos que están en el mejor camino hacia una solución óptima. Por ejemplo, en el problema del mundo de bloques de la Figura 8.2, una buena estrategia de control debería indicar que el estado ((A)(B C)) es más prometedor que el estado ((A B)(C)). Sin embargo, una estrategia de búsqueda no informada, o búsqueda a ciegas, considerará a estos dos nodos, y a los otros cuatro sucesores del estado *inicial*, igualmente prometedores. Como resultado, lo que cabe esperar es que una búsqueda a ciegas requiera visitar una cantidad de estados mucho mayor que una búsqueda inteligente para encontrar una solución del problema. Por lo tanto, lo que se pretende con un sistema de búsqueda inteligente es encontrar una buena solución del problema, a ser posible óptima, visitando la menor cantidad de estados posible. Entre las diversas definiciones que se han propuesto para el término IA, alguna de ellas pone el énfasis precisamente en este aspecto. Por ejemplo la definición propuesta por D. Patridge, y que se puede ver en [Russell y Norvig, 2003]: *La IA adecuadas para problemas específicos*.

El resto de este capítulo se dedica al estudio de los métodos de búsqueda a ciegas y algunas de sus propiedades principales, como por ejemplo la cantidad de recursos computacionales que requieren y sus propiedades de completitud y admisibilidad, que se definen del siguiente modo

Definición 8.1. *Un algoritmo de búsqueda es completo si siempre encuentra solución, en el caso de que exista alguna.*

Definición 8.2. *Un algoritmo de búsqueda es admisible, o exacto, si siempre encuentra una solución óptima.*

8.4 Métodos de búsqueda sin información

Como ya se ha comentado, este capítulo describe los métodos básicos de búsqueda sin información, es decir métodos que realizan un recorrido del espacio de búsqueda de una forma sistemática, pero sin tener en cuenta ningún tipo de información sobre el dominio del problema que se está resolviendo. Para facilitar la exposición, se considera en principio que el espacio de búsqueda es un árbol, lo cual simplifica bastante las

operaciones. Luego se tendrán en cuenta las modificaciones que hay que introducir para tratar con espacios de búsqueda más generales, es decir grafos.

En la descripción de los algoritmos se utilizará una notación y un nivel de detalle, en las operaciones y estructuras de datos, que facilite la implementación de los mismos. En principio los algoritmos serán iterativos. Las estructuras de datos y operaciones principales serán las que se indican a continuación. Se utiliza una lista denominada *ABIERTA* que contendrá al principio de cada iteración los estados candidatos a ser desarrollados o expandidos y que estará ordenada con un determinado criterio en cada caso. El desarrollo, o expansión de un estado, consiste en calcular todos sus sucesores mediante la aplicación de todos los operadores posibles, para ello se dispone de la función *Sucesores* que para un determinado estado retorna una lista con todos sus sucesores. Se utiliza otra estructura denominada *TABLA_A*, que es en realidad un conjunto ordenado y se suele implementar mediante una tabla hash, para registrar toda la información necesaria acerca de cada uno de los nodos encontrados en el proceso de búsqueda, por ejemplo el padre de cada uno de los nodos, o el coste desde el estado *inicial* hasta cada nodo. El resto de las operaciones y estructuras tienen un significado natural.

8.4.1 Recorrido de árboles

El algoritmo 8.1 muestra un esquema genérico de búsqueda sin información en árboles. En primer lugar se coloca en la lista *ABIERTA* el estado *inicial*. Este estado se genera a partir de los datos del problema. Por ejemplo, en el problema del TSP será una secuencia que solamente incluye a la ciudad de partida. A partir de ese momento el algoritmo va expandiendo nodos, de forma que en cada iteración el nodo que se expande es el primero de la lista *ABIERTA*. De este modo, la estrategia de control viene definida por el criterio que se utilice para ordenar *ABIERTA*, o lo que es igual por el criterio que se utilice para insertar los nodos en esta lista. En las secciones siguientes se muestran las tres formas más comunes de realizar una búsqueda a ciegas y se analiza su comportamiento en términos del espacio y el tiempo requeridos para una búsqueda, así como de sus propiedades de completitud y admisibilidad.

8.4.1.1 Primero en Anchura

La búsqueda en anchura se obtiene a partir del algoritmo 8.1 sin más que insertar los sucesores siempre al final de *ABIERTA*, con lo que *ABIERTA* se trata realmente como una cola. Este algoritmo es completo y, en el caso de que todas las reglas tengan coste igual a 1, es también admisible. Sin embargo, tanto el tiempo de ejecución como el espacio de memoria necesario, crecen de forma exponencial con el tamaño del problema. Para justificar esto último supóngase un factor de ramificación, o número medio de sucesores, b y que hay una única solución a una profundidad d , siendo la profundidad de un nodo el número de arcos que hay en el camino desde el *inicial* hasta el nodo. El número de nodos visitados se puede calcular de la siguiente forma.

Algoritmo 8.1 Algoritmo de búsqueda sin información en árboles.

```

1: ABIERTA = (inicial);
2: mientras NoVacia(ABIERTA) hacer
3:   n = ExtraePrimero(ABIERTA);
4:   si EsObjetivo(n) entonces
5:     devolver Camino(inicial, n); {leyéndolo en la TABLA_A}
6:   fin si
7:   S = Sucesores(n);
8:   para cada q de S hacer
9:     pone q en la TABLA_A con
       Anterior(q) = n,
       Coste(inicial, q) = Coste(inicial, n) + Coste(n, q);
10:    inserta q en ABIERTA;
11:   fin para
12: fin mientras
13: devolver "no encontrado";

```

EL número de nodos hasta el nivel $d - 1$ es:

$$1 + b + b^2 + \cdots + b^{d-1} = \frac{b^d - 1}{b - 1} \quad (8.1)$$

Si además se han visitado la mitad de los nodos del nivel d , se debe añadir a la cantidad anterior el valor

$$\frac{b^d}{2} \quad (8.2)$$

Luego el número total de nodos visitados, para un d grande, se puede aproximar por

$$\frac{b^{d+1} + b^d}{2(b - 1)} \quad (8.3)$$

En cuanto al espacio requerido por las estructuras *ABIERTA* y *TABLA_A* se puede aproximar del siguiente modo. La lista *ABIERTA* tendrá la mitad de los nodos del nivel d y la mitad de los del nivel $d - 1$, es decir

$$\frac{b^d + b^{d-1}}{2} \quad (8.4)$$

Y la *TABLA_A* tendrá todos los nodos hasta el nivel $d - 2$ y la mitad de los del nivel $d - 1$, es decir, aproximadamente

$$\frac{b^d + b^{d-1}}{2(b - 1)} \quad (8.5)$$

Luego son precisas cantidades de tiempo y de espacio que son exponenciales en la profundidad d de la solución.

8.4.1.2 Primero en Profundidad

La estrategia de primero en profundidad se obtiene también, en principio, como un caso particular del algoritmo 8.1. En este caso los nodos se insertan al principio de *ABIERTA*, con lo cual esta estructura se trata como una pila. A diferencia de la estrategia de primero en anchura, la estrategia de primero en profundidad no es admisible ni siquiera completa. Esto último es debido a que la búsqueda se puede derivar hacia una rama infinita, con lo que el algoritmo no termina. Para evitar este inconveniente, en problemas con espacios de búsqueda infinitos, se suele establecer una profundidad límite a partir de la cual se detiene la búsqueda. En este caso el algoritmo termina, pero tampoco hay garantía de que sea completo ya que la solución puede encontrarse más allá de la profundidad límite elegida. El algoritmo 8.2 muestra un esquema del algoritmo de búsqueda en profundidad en el que se ha añadido el control de la profundidad límite y la operación *LimpiarTABLA_A* que permite eliminar de la *TABLA_A* aquellos estados que ya no son necesarios. Cuando esta operación se aplica a un estado n , lo que hace es simplemente eliminar de la *TABLA_A* al padre p de n siempre y cuando p no tenga sucesores en *ABIERTA*. La operación se aplica recursivamente al nodo p si este nodo es realmente eliminado de la *TABLA_A*. No se trata de una operación compleja ya que en el caso de que el nodo p tenga sucesores en *ABIERTA*, estos sucesores serán siempre los primeros nodos de esta lista. De este modo la *TABLA_A* contiene únicamente información de los caminos desde el estado *inicial* hasta aquellos que están en *ABIERTA*.

Algoritmo 8.2 Algoritmo de búsqueda primero en profundidad en árboles.

```
1: ABIERTA = (inicial);
2: mientras NoVacia(ABIERTA) hacer
3:    $n = \text{ExtraePrimero}(\text{ABIERTA})$ ;
4:   si EsObjetivo(n) entonces
5:     devolver Camino(inicial, n);
6:   fin si
7:    $S = \emptyset$ ;
8:   si Profundidad(n) < ProfundidadLímite entonces
9:      $S = \text{Sucesores}(n)$ ;
10:   fin si
11:   si Vacia(S) entonces
12:     LimpiarTABLA_A(n);
13:   fin si
14:   para cada  $q$  de  $S$  hacer
15:     pone  $q$  en la TABLA_A con
         $\text{Anterior}(q) = n$ ,
         $\text{Coste}(\text{inicial}, q) = \text{Coste}(\text{inicial}, n) + \text{Coste}(n, q)$ ,
         $\text{Profundidad}(q) = \text{Profundidad}(n) + 1$ ;
16:     inserta  $q$  al principio de ABIERTA;
17:   fin para
18: fin mientras
19: devolver "no encontrado";
```

Si la profundidad límite es igual a d , el espacio máximo requerido para *ABIERTA* es $(b-1)(d-2)+b$; y $1+(d-1)b$ para la *TABLA_A*. Luego el espacio es proporcional a la profundidad d . Si la solución está a profundidad d , el tiempo de búsqueda se puede estimar del siguiente modo, en función de que el nodo solución sea el primero a la izquierda, el primero a la derecha o el nodo intermedio. Si es el primero a la izquierda, el número de nodos visitados es $(d+1)$. Si es el primero a la derecha, habrá que visitar todos los nodos hasta la profundidad d , luego el tiempo será proporcional a

$$1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1} \quad (8.6)$$

Y si se trata del nodo intermedio del nivel d el tiempo será proporcional a

$$\frac{b^{d+1} - 1}{2(b - 1)} \quad (8.7)$$

Es decir, el tiempo de búsqueda es exponencial en la profundidad límite.

Al comparar las estrategias de búsqueda en anchura y búsqueda en profundidad, se puede ver que, con respecto al espacio de memoria, la primera requiere una cantidad exponencial en la profundidad del árbol, mientras que la segunda requiere una cantidad lineal. Sin embargo, las dos requieren un tiempo exponencial en la profundidad del árbol. Pero en este caso, para un valor grande de la profundidad, si el objetivo es un nodo intermedio del nivel d , la búsqueda en profundidad es mejor en un factor $(1 + 1/b)$ que la búsqueda en anchura.

Por otra parte, la búsqueda en anchura siempre proporciona la solución más cercana al nodo *inicial*, lo que no ocurre con la búsqueda en profundidad. Además, la búsqueda en profundidad requiere establecer una profundidad límite para evitar el inconveniente de las ramas infinitas.

Como alternativa al algoritmo 8.2, la búsqueda en profundidad se suele realizar mediante algoritmos de tipo backtracking. Se trata de algoritmos recursivos que resultan más simples y elegantes debido a que no requieren el uso de estructuras como *ABIERTA* y la *TABLA_A*, ya que su función la realiza la propia pila que gestiona las llamadas recursivas. Otra diferencia es que en lugar de utilizar la función *Sucesores* para aplicar todas las reglas posibles a un estado, éstas se aplican de una en una, con el correspondiente ahorro de tiempo y espacio. El algoritmo 8.3 muestra el esquema genérico del backtracking. Este algoritmo se aplica a un estado n y produce *FALLO* si no hay solución alcanzable desde este estado, o bien una lista de reglas que permite alcanzar un objetivo desde el estado n .

En la práctica, los algoritmos de backtracking son muy utilizados debido al bajo consumo de espacio de almacenamiento. Además, para mejorar su eficiencia se suelen combinar con algún tipo de conocimiento sobre el dominio del problema concreto. Este conocimiento se puede introducir en el cálculo de las reglas aplicables a cada estado y en la ordenación de estas reglas. Esto es lo que hacen, por ejemplo, los conocidos heurísticos de ordenación de variables y valores que se utilizan con frecuencia para resolver problemas de satisfacción de restricciones (véase el capítulo 10).

Algoritmo 8.3 Algoritmo de backtracking.

```
1: Backtracking( $n$  : estado) : lista de reglas;
2: si EsObjetivo( $n$ ) entonces
3:   devolver NIL;
4: fin si
5: si EsSituacionSinSalida( $n$ ) entonces
6:   devolver FALLO;
7: fin si
8: si Profundidad( $n$ )  $\geq$  ProfundidadLímite entonces
9:   devolver FALLO;
10: fin si
11: ListaR = lista de reglas aplicables a  $n$ ;
12: mientras NoVacia(ListaR) hacer
13:    $R = \text{ExtraePrimero}(\text{ListaR})$ ;
14:    $q =$  resultado de aplicar  $R$  a  $n$ ;
15:   Profundidad( $q$ ) = Profundidad( $n$ ) + 1;
16:    $L = \text{Backtracking}(q)$ ;
17:   si  $L \neq$  FALLO entonces
18:     devolver push( $R, L$ );
19:   fin si
20: fin mientras
21: devolver FALLO;
```

8.4.1.3 Coste uniforme

Si en lugar de tratar *ABIERTA* como una pila o una cola, como se hace en las estrategias anteriores de búsqueda en profundidad y búsqueda en anchura, se insertan los nodos de forma ordenada teniendo en cuenta el coste desde el nodo *inicial* a cada uno de los nodos, se obtiene la estrategia del coste uniforme. Se trata de una estrategia similar a la búsqueda en anchura: de hecho, si el coste de todas las reglas es el mismo, las dos estrategias son equivalentes. El coste computacional en tiempo y espacio es el mismo que en la búsqueda en anchura, pero la solución que calcula la estrategia del coste uniforme es la de menor coste al *inicial*, con lo que el algoritmo es admisible.

8.4.1.4 Búsquedas en profundidad y en anchura iterativas

Una forma de resolver el problema de la profundidad límite de la búsqueda en profundidad podría ser considerar de forma iterativa sucesivos valores de este parámetro, y para cada uno de ellos realizar una búsqueda completa. Esta es la estrategia denominada búsqueda en profundidad iterativa. El principal inconveniente de este tipo de búsqueda es que para cada uno de los valores de la profundidad límite hay que visitar de nuevo todos los visitados en la iteración anterior. Sin embargo en la práctica esto no supone un inconveniente muy grave ya que, el número de nodos nuevos es en realidad mucho mayor que los visitados en las iteraciones anteriores. Al igual que en la búsqueda en profundidad convencional, si la profundidad máxima alcanzada es d , el espacio de memoria es proporcional a d , y el tiempo de búsqueda es exponencial en d . El algoritmo 8.4 muestra una versión del algoritmo de búsqueda en profundidad iterativa que se obtiene haciendo variar la profundidad límite a partir de 1, aplicando el algoritmo convencional de búsqueda en profundidad para cada uno de estos valores.

Algoritmo 8.4 Algoritmo de búsqueda primero en profundidad iterativa en árboles.

```

1: ProfundidadLimite = 1;
2: Solucion = BusquedaPrimeroenProfundidad;
3: mientras Solucion = “no encontrado” hacer
4:   ProfundidadLimite = ProfundidadLimite + 1;
5:   Solucion = BusquedaPrimeroenProfundidad;
6: fin mientras
7: devolver Solucion;

```

De forma similar, se puede diseñar un algoritmo de búsqueda en anchura iterativa. En este caso el parámetro *AnchuraLimite* permite controlar el número máximo de sucesores de cada estado. El algoritmo 8.5 muestra una versión de este método. La operación *Sucesores(n, AnchuraLimite)* selecciona el número de sucesores que se consideran en cada iteración, de acuerdo con el parámetro *AnchuraLimite*. La iteración puede continuar mientras esta operación haya dejado fuera a alguno de los sucesores de algún nodo expandido.

Como en el caso de la búsqueda en anchura convencional este algoritmo requiere un espacio y un tiempo que son exponentiales en la profundidad del árbol de búsqueda. Sin embargo, la búsqueda primero en anchura iterativa puede encontrar una solución que no sea la más próxima al estado *inicial*.

Algoritmo 8.5 Algoritmo de búsqueda primero en anchura iterativa en árboles.

```

1: AnchuraLimite = 0;
2: mientras se pueda aumentar la AnchuraLimite hacer
3:   AnchuraLimite = AnchuraLimite + 1;
4:   ABIERTA = (inicial);
5:   mientras NoVacia(ABIERTA) hacer
6:     n = ExtraePrimero(ABIERTA);
7:     si EsObjetivo(n) entonces
8:       devolver Camino(inicial, n);
9:     fin si
10:    S = Sucesores(n, AnchuraLimite);
11:    para cada q de S hacer
12:      poner q en la TABLA_A con
13:        Anterior(q) = n,
14:        Coste(inicial, q) = Coste(inicial, n) + Coste(n, q);
15:      insertar q al final de ABIERTA;
16:    fin para
17:  fin mientras
18: fin mientras
19: devolver “no encontrado”;

```

8.4.2 Recorrido de grafos

En las secciones anteriores se ha considerado que el espacio de búsqueda era un árbol. Esto es cierto en muchos casos, por ejemplo en el problema del viajante de comercio, pero no lo es en muchos otros, como por ejemplo en el cálculo de planes de actuación de robots. Si el espacio de búsqueda es un grafo, hay que tener en cuenta que

durante la búsqueda se puede encontrar varias veces el mismo estado n como sucesor de dos estados diferentes. En principio se pueden tratar estas dos apariciones de n como estados distintos y continuar la búsqueda como si se tratase de un árbol. Sin embargo esto puede ser muy ineficiente debido a que se estaría recorriendo varias veces el espacio a partir de este estado n . En la práctica suele ser mejor considerar los distintos caminos que se encuentran hasta el estado n y registrar el mejor obtenido hasta el momento. Para ello hay que rectificar el contenido almacenado en la *TABLA_A* para el nodo n . En el caso de encontrar un camino mejor desde el *inicial* hasta n , basta con modificar el nodo anterior en la *TABLA_A* y el coste desde el *inicial* hasta n . Previamente hay que comprobar si cada uno de los sucesores q del estado n que se expande es un nodo que aparece por primera vez, o bien ya apareció previamente, lo que supone un coste añadido con respecto a la búsqueda en árboles.

Como método de recorrido de grafos se presenta a continuación el algoritmo general de búsqueda en grafos propuesto por N. J. Nilsson [Nilsson, 2001], [Nilsson, 1998], [Nilsson, 1987], [Nilsson, 1971] aunque su principal difusor fue seguramente J. Pearl [Pearl, 1984]. Este algoritmo es una generalización de los algoritmos anteriores de búsqueda sin información y, como se verá en el capítulo siguiente, se puede particularizar también para realizar distintos tipos de búsqueda inteligente.

8.4.2.1 El algoritmo general de búsqueda en grafos

El Algoritmo General de Búsqueda en Grafos (AGBG) parte de un grafo dirigido simple definido implícitamente a partir de un nodo inicial y una serie de operadores o reglas de producción. Cada regla tiene un coste no negativo, concretamente debe existir un valor real $\varepsilon > 0$ tal que el número de reglas con un coste menor o igual que ε sea finito. De este modo no existen caminos en el espacio de búsqueda con un número infinito de nodos y un coste acotado. Además, el grafo debe ser localmente finito, es decir que cada nodo o estado tiene un número finito de sucesores, aunque no necesariamente finito. En el grafo hay uno o varios estados solución y el objetivo de la búsqueda es encontrar el camino de coste mínimo desde el nodo inicial a los estados objetivo.

Para realizar las operaciones de rectificación de los nodos es necesario registrar explícitamente todo el espacio de búsqueda desarrollado hasta el momento. Para ello, se añade un campo adicional a la entrada de cada nodo n en la *TABLA_A*. Esta entrada registra la lista de sucesores de n , así como los costes desde n a cada uno de ellos, para cada nodo n expandido. Si el nodo n no ha sido aún expandido, esta lista será nula.

El algoritmo 8.6 muestra un esquema del AGBG. El algoritmo comienza colocando el estado *inicial* en la lista *ABIERTA* y a partir de esa situación, al igual que los algoritmos de búsqueda a ciegas de las secciones anteriores, va expandiendo nodos de forma iterativa hasta que *ABIERTA* se agote, en cuyo caso no hay solución, o bien hasta encontrar un nodo solución, en cuyo caso devuelve la secuencia de nodos desde el estado *inicial* hasta el objetivo encontrado. Esta secuencia se obtiene encadenando entradas de la *TABLA_A*. Una vez expandido un nodo n , el primer paso es registrar sus sucesores en la entrada de n en la *TABLA_A*, para permitir la rectificación

posterior de los sucesores de n si es preciso. A continuación, se analiza cada uno de los sucesores q del nodo n . Al ser el espacio de búsqueda un grafo, es posible que un sucesor q sea un nodo ya encontrado previamente, en consecuencia estará registrado en la $TABLA_A$ y también en $ABIERTA$ si aún no fue expandido. Si el nodo q ya está en la $TABLA_A$ hay que comprobar si el nuevo camino, a través del nodo n que se acaba de expandir, es mejor que el camino que está registrado en la $TABLA_A$. Si el nuevo camino es mejor, hay que actualizar la información en la entrada correspondiente al nodo q . Esto es lo que hace la función *Rectificar* que se muestra en el algoritmo 8.7. Además, si el nodo q ya fue expandido y el nuevo camino a través de n es mejor que el registrado anteriormente, es posible que se haya encontrado también un camino mejor para alguno de los sucesores de q . Para comprobar esto se utiliza la función *RectificarLista*, que se muestra también en el algoritmo 8.7. La función *RectificarLista* obtiene de la $TABLA_A$ los sucesores de un nodo q ya expandido y aplica la función *Rectificar* a cada uno de ellos, con lo que el proceso se puede repetir de forma recursiva hasta llegar a nodos que se encuentren en $ABIERTA$, y que por lo tanto no tienen sucesores registrados en la $TABLA_A$. De esta forma, la $TABLA_A$ registra el mejor camino obtenido hasta el momento desde el nodo *inicial* a cada uno de los nodos encontrados durante la búsqueda.

Algoritmo 8.6 Algoritmo general de búsqueda en grafos.

```

1:  $ABIERTA = (\text{inicial});$ 
2: mientras  $NoVacia(ABIERTA)$  hacer
3:    $n = ExtraePrimero(ABIERTA);$ 
4:   si  $EsObjetivo(n)$  entonces
5:     devolver  $Camino(\text{inicial}, n)$  ;
6:   fin si
7:    $S = Sucesores(n);$ 
8:   Añade  $S$  a la entrada de  $n$  en la  $TABLA\_A$ ;
9:   para cada  $q$  de  $S$  hacer
10:    si  $(q \in TABLA\_A)$  entonces
11:       $Rectificar(q, n, Coste(n, q));$ 
12:       $Ordenar(ABIERTA); \{si\ es\ preciso\}$ 
13:    si no
14:      pone  $q$  en la  $TABLA\_A$  con
           $Anterior(q) = n,$ 
           $Coste(\text{inicial}, q) = Coste(\text{inicial}, n) + Coste(n, q);$ 
15:       $ABIERTA = Mezclar(q, ABIERTA);$ 
16:    fin si
17:   fin para
18: fin mientras
19: devolver “no solución”;

```

Si el nodo q sucesor de n es un nodo nuevo, se crea una entrada para q en la $TABLA_A$ en la que se registra el primer camino encontrado para él desde el *inicial*. Además, el nodo q se inserta en $ABIERTA$ mediante la operación genérica $Mezclar(q, ABIERTA)$. La estrategia de control del AGBG viene definida por el criterio de ordenación de la lista $ABIERTA$ que está implementado en esta operación y también en la operación $Ordenar(ABIERTA)$. Esta última operación se utiliza después del proceso de rectificación de los nodos debido a que normalmente el criterio

de ordenación de *ABIERTA* depende, entre otros factores, del coste del mejor camino obtenido hasta el momento desde el inicial a cada uno de los nodos de *ABIERTA*. En esta ordenación solamente habrá que reubicar, como mucho, a aquellos nodos cuyo camino al *inicial* haya mejorado.

Algoritmo 8.7 Funciones de rectificación de nodos ya expandidos.

```
Función Rectificar( $n, p, costepn$ );
    si ( $Coste(inicial, p) + costepn < Coste(inicial, n)$ ) entonces
        Modifica la entrada del nodo  $n$  en la TABLA_A con
             $Coste(inicial, n) = Coste(inicial, p) + costepn;$ 
             $Anterior(n) = p;$ 
            RectificarLista( $n$ );
    fin si
```

```
Función RectificarLista( $n$ );
    LISTA = Sucesores( $n$ ); /* Registrados en la TABLA_A */
    para cada  $q$  de LISTA hacer
        Rectificar( $q, n, Coste(n, q)$ );
    fin para
```

El ejemplo de la Figura 8.4(a) muestra una situación en la que se han desarrollado tres nodos, el *inicial*, $n1$ y $n2$, y en la que hay tres nodos en *ABIERTA*, $n3$, $n4$ y $n5$. Como se puede observar, ya han sido encontrados dos caminos distintos desde el nodo *inicial* al nodo $n4$, por lo que solamente se registra el mejor hasta el momento en la *TABLA_A*. El contenido de esta tabla para la situación de la Figura 8.4 se muestra en la Tabla 8.1. La Figura 8.4(b) muestra la situación resultante de la expansión del nodo $n3$. Solamente uno de los sucesores de este nodo es nuevo, el $n6$, otro, el $n2$, ya ha sido desarrollado y el tercero, $n5$, se encuentra en *ABIERTA*. En consecuencia el nodo $n6$ se inserta en *ABIERTA* y en la *TABLA_A*, mientras que los otros dos sufren el proceso de rectificación. La situación resultante de la *TABLA_A* se muestra también en la Tabla 8.1.

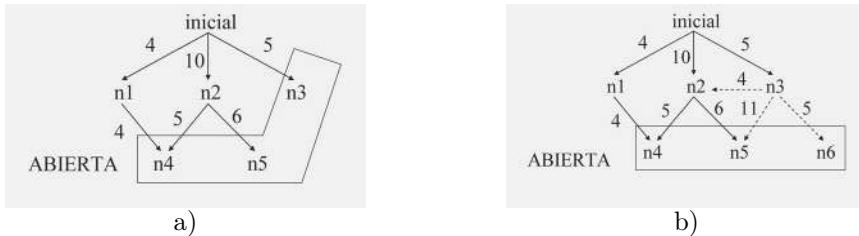


Figura 8.4: Ejemplo de aplicación del AGBG. Espacio de búsqueda desarrollado antes de la expansión del nodo $n3$ (a) y después de la expansión de este nodo (b).

Las estrategias clásicas de búsqueda a ciegas en grafos se pueden obtener como casos particulares del AGBG. Concretamente si se prescinde de la operación *Ordenar* y la operación *Mezcla* inserta los nodos al principio de *ABIERTA*, entonces se tiene la estrategia de búsqueda en profundidad; y si la operación *Mezcla* inserta los nodos

al final de *ABIERTA*, resulta la estrategia de búsqueda en anchura. Si simplemente se mantiene como criterio de ordenación de *ABIERTA* el coste al inicial, de menor a mayor, la estrategia resultante es la del coste uniforme.

Clave	Anterior	Coste al Inicial	Sucesores
Contenido antes de la expansión del nodo n_3			
<i>inicial</i>	-	0	$((n_1\ 4)\ (n_2\ 10)\ (n_3\ 5))$
n_1	<i>inicial</i>	6	$((n_4\ 4))$
n_2	<i>inicial</i>	10	$((n_4\ 5)\ (n_5\ 6))$
n_3	<i>inicial</i>	5	-
n_4	n_1	8	-
n_5	n_2	16	-
Contenido después de la expansión del nodo n_3			
<i>inicial</i>	-	0	$((n_1\ 4)\ (n_2\ 10)\ (n_3\ 5))$
n_1	<i>inicial</i>	6	$((n_4\ 4))$
n_2	n_3	9	$((n_4\ 5)\ (n_5\ 6))$
n_3	<i>inicial</i>	5	$((n_2\ 4)\ (n_5\ 11)\ (n_6\ 5))$
n_4	n_1	8	-
n_5	n_2	15	-
n_6	n_3	10	-

Tabla 8.1: Contenido de la *TABLA_A* para las dos situaciones de la Figura 8.4.

La descripción que se ha hecho aquí del AGBG difiere en algunos aspectos de las versiones de este algoritmo que suelen presentar los textos clásicos, como por ejemplo los textos de N. J Nilsson [Nilsson, 2001] y J. Pearl [Pearl, 1984]. Estas versiones no realizan el proceso de rectificación del mismo modo en el que se ha presentado aquí, sino que utilizan una estructura similar a la *TABLA_A* solamente para registrar el mejor camino desde el *inicial* a cada nodo, y además una lista *CERRADA* en la que se guardan los nodos expandidos. Cuando un nodo ya expandido sufre una rectificación, este nodo se mueve de *CERRADA* a *ABIERTA* para ser expandido de nuevo. Esta es una alternativa al proceso que realiza la operación *RectificarLista* que puede resultar más o menos eficiente dependiendo de cada caso concreto. Si esta operación se realiza pocas veces, puede ser más eficiente volver a expandir nodos que mantener todo el grafo explorado en la *TABLA_A* mediante el registro de los sucesores de cada nodo expandido.

8.4.2.2 Búsqueda bidireccional

La idea de la búsqueda bidireccional consiste en buscar simultáneamente en las dos direcciones, es decir, hacia delante desde el *inicial* a los objetivos y hacia atrás desde los objetivos al *inicial*. La búsqueda se detiene cuando las correspondiente fronteras, es decir las dos listas *ABIERTA*, tienen algún nodo en común. De este modo si hay una solución a profundidad d , entonces cada uno de los dos algoritmos tiene que buscar hasta una profundidad $d/2$, con lo que idealmente el tiempo de ejecución será

exponencial en $d/2$ en lugar de serlo en d . Lo que supone una mejora realmente importante.

Sin embargo, para aplicar la búsqueda bidireccional hay que tener en cuenta varios factores que condicionan esta posibilidad. El más importante es cuántos y cómo son los estados objetivos. En principio la búsqueda hacia atrás se debe realizar desde cada uno de los estados objetivo, con lo cual si son muchos la suposición anterior sobre la complejidad de la búsqueda dejaría de ser cierta, como ocurre por ejemplo en el problema del viajante de comercio. Además, en muchos casos, no es fácil conocer a priori la forma de los estados objetivo. Otra consideración importante es si los operadores son realmente bidireccionales y si el factor de ramificación es el mismo en los dos sentidos de la búsqueda. Por ejemplo, en el problema de la generación de planes de actuación de robots esto es realmente así, pero en otros problemas no resulta natural decidir cuáles son los sucesores de un estado en la búsqueda hacia atrás. Hay que decidir también qué tipo de búsqueda aplicar en cada dirección. Para que las dos búsquedas se encuentren en algún momento, al menos una de las dos debe retener en memoria todos los estados visitados, es decir debe ser una búsqueda en anchura, con lo que el espacio requerido será exponencial en $d/2$. Por último, hay que tener una operación eficiente para verificar la intersección de las dos listas *ABIERTA*. Esto se puede realizar en tiempo constante con algún método avanzado de representación de conjuntos ordenados, por ejemplo tablas hash.

8.5 Resumen

En este capítulo se han introducido los sistemas de búsqueda, o sistemas de resolución de problemas, en el campo de la IA. Se ha visto que un sistema de búsqueda tiene dos componentes principales que son el espacio de búsqueda y la estrategia de control, o algoritmo de búsqueda. El capítulo se ha centrado en el paradigma de búsqueda en espacios de estados porque se trata del método de aplicación más general. En este caso el espacio de búsqueda tiene forma de grafo dirigido simple. Como estrategia de control se han descrito varios algoritmos, en particular el Algoritmo General de Búsqueda en Grafos propuesto por N. J. Nilsson, y cómo este algoritmo se puede particularizar para realizar distintos tipos de búsqueda en anchura y en profundidad, sin utilizar ningún tipo de información sobre el dominio del problema, lo que se suele denominar búsqueda no informada o búsqueda a ciegas. También se han estudiado los requerimientos computacionales, de espacio y tiempo, de algunas de las estrategias descritas; así como sus propiedades de completitud y admisibilidad. Además se ha hecho hincapié en los pasos que hay que seguir para resolver un problema, que básicamente consisten en modelar los estados y los operadores, de acuerdo con las capacidades de percepción y actuación del agente que tiene que aplicar la solución del problema, y luego elegir un algoritmo de búsqueda para calcular un camino desde el estado inicial a uno de los objetivos. Todos estos pasos se han ilustrado con algunos ejemplos de problemas formales como son el problema de las N -reinas, el TSP, el 8-puzzle o un problema de scheduling. En la sección siguiente se propone la realización de algunos ejercicios adicionales de características similares a los anteriores. Como

complemento a estos ejercicios, en el texto de S. Fernández, J. González, y J. Mira [Fernández y otros, 1998] el lector interesado puede encontrar una gran variedad de problemas resueltos con distintas técnicas de búsqueda.

Este capítulo sirve como introducción del capítulo 9 que tiene como objetivo introducir las técnicas de búsqueda inteligente o búsqueda informada, donde se verá cómo incorporar, en la estrategia de búsqueda, cualquier tipo de información o conocimiento, más o menos preciso, sobre el dominio del problema que se pretende resolver. El objetivo será guiar la búsqueda hacia los caminos más prometedores y conseguir así sistemas más eficientes que encuentren buenas soluciones con un coste computacional mucho menor que aquel que normalmente requiere un algoritmo de búsqueda a ciegas.

8.6 Ejercicios resueltos

En esta sección se presentan algunos ejemplos de aplicación de algoritmos de búsqueda a la resolución de problemas. Como se ha comentado en la sección 8.3, para resolver un problema con un algoritmo de búsqueda en primer lugar hay que describir el espacio de búsqueda, para ello hay que definir un único estado inicial y un conjunto de operadores que se irán aplicando a partir de este estado. Cada operador debe quedar perfectamente especificado mediante una descripción precisa de los estados a los que se puede aplicar, y en cada caso de una descripción precisa del estado resultante. La aplicación de un operador a un estado tiene un coste no negativo. En los algoritmos anteriores, la función *Sucesores* es la responsable de aplicar a un estado n todos los operadores posibles, así como de calcular la correspondiente lista de sucesores y los costes desde n a cada sucesor. También hay que definir una función que permita determinar si un estado es objetivo. El modelo elegido para la representación de los estados y de los operadores debe permitir expresar, con un nivel de detalle adecuado, las situaciones del entorno real que puede reconocer el agente, así como las operaciones elementales que es capaz de realizar sobre el entorno. El espacio de búsqueda definido debe ser completo, es decir debe permitir alcanzar al menos una de las soluciones óptimas. Además, por razones de eficiencia, el espacio de búsqueda debe ser lo más reducido posible. En los ejemplos se verá que en ocasiones es posible evitar caminos redundantes entre dos nodos mediante una selección cuidadosa de los sucesores de cada estado. Si además el espacio de búsqueda es un árbol, la búsqueda será en general más eficiente. Una vez definido el espacio de búsqueda, se podrá aplicar cualquier estrategia de control.

8.1. El problema de las N -reinas. Se trata de un problema muy conocido en la literatura de búsqueda que consiste en colocar N reinas sobre un tablero de $N \times N$ casillas de forma que ningún par de reinas se ataquen mutuamente, es decir que no estén situadas ni en la misma fila, ni en la misma columna, ni en la misma diagonal. La Figura 8.5 muestra una solución para el problema de las 4 reinas. En este pequeño ejemplo es fácil ver que ésta es la única solución posible, salvo simetrías.

La representación de los estados en este caso es bastante evidente. Lo natural, es interpretar cada estado como una situación del tablero en la que hay un subconjunto

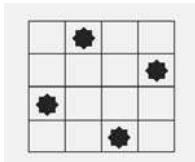


Figura 8.5: Una solución del problema de las 4 reinas.

de las N reinas de forma que ningún par de ellas se ataquen entre sí. El estado inicial será el tablero vacío, y cualquier estado con N reinas será un objetivo. La Figura 8.6 muestra varios estados posibles. Con esta interpretación de los estados, los operadores se pueden interpretar de varias formas diferentes. En este caso lo natural es que la aplicación de un operador consista en colocar una reina en el tablero (aunque también podríamos considerar la colocación de varias) y que todas las reglas tengan un coste igual a 1, con lo que el coste de todas las soluciones es el mismo, pues en todas hay que colocar N reinas. Como primera opción se puede considerar que dado un estado hay tantos sucesores como casillas quedan en el tablero que no son “atacadas” por ninguna reina. Así, el estado inicial tendrá $N \times N$ sucesores, y claramente el espacio de búsqueda será un grafo. También será un espacio de búsqueda completo, ya que en realidad representa todas las posibilidades que hay para llegar a una solución colocando las reinas de una en una. Sin embargo, esta no parece ser la mejor opción posible.

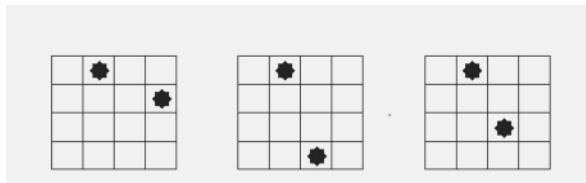


Figura 8.6: Tres estados válidos para el problema de las 4 reinas.

Otra posibilidad para definir el espacio de búsqueda, concretamente los operadores, consiste en ir colocando las reinas de forma más ordenada; por ejemplo, la primera reina en la primera fila, la segunda reina en la segunda fila, y así sucesivamente. De este modo también se obtiene un espacio completo, ya que cada reina en la solución debe estar en una fila distinta. Además, el espacio de búsqueda es más reducido, ya que no contiene a un gran número de los estados que sí contiene el espacio anterior, como por ejemplo el segundo y el tercero de los que se muestran en la Figura 8.6. Además, el espacio así definido es un árbol. Si se tiene en cuenta la simetría de las soluciones, se pueden descartar la mitad de los sucesores del estado *inicial*. De este modo, la segunda opción es claramente mejor que la primera. La Figura 8.7 muestra el espacio de búsqueda completo para el problema de las 4 reinas.

En el capítulo 10 se verán otras técnicas aplicables a este problema bajo su planteamiento como problema de satisfacción de restricciones.

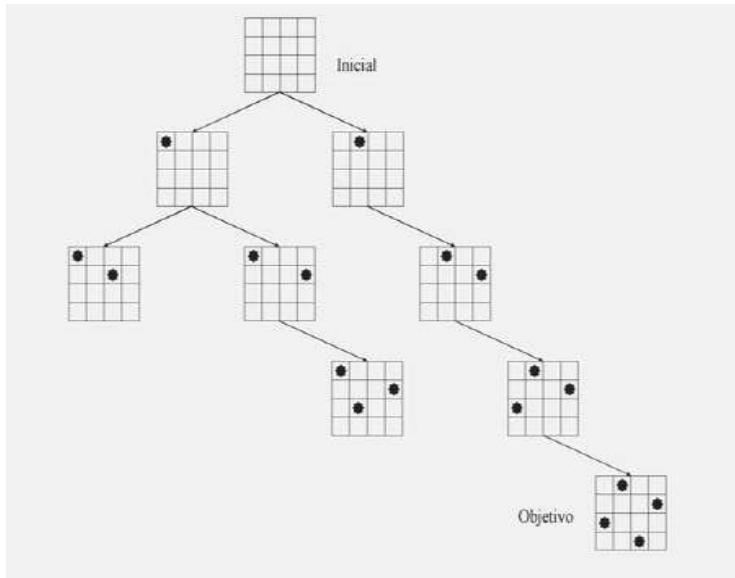


Figura 8.7: Espacio de búsqueda para el problema de las 4 reinas.

■

8.2. El problema del viajante de comercio. Este problema se conoce normalmente por sus siglas en inglés TSP (Traveling Salesman Problem) y es uno de los problemas más famosos en optimización combinatoria. El enunciado es el siguiente: dado un conjunto de N ciudades y los costes de conexión entre cada par de ellas, se trata de encontrar una ruta de coste mínimo que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades. Es decir, se trata de encontrar una permutación del conjunto de ciudades $P = \{c_0, \dots, c_{N-1}\}$ que minimice la función

$$d_P = \sum_{i=0}^{N-1} d(c_i, c_{(i+1) \mod N}) \quad (8.8)$$

donde $d(x, y)$ es la distancia de la ciudad x a la ciudad y .

La formulación más simple del TSP como un problema de búsqueda en espacios de estados consiste en interpretar cada estado como una subsecuencia de ciudades visitadas, de modo que el estado *inicial* incluye solamente a la ciudad de partida y los estados objetivos vienen dados por una secuencia formada por la ciudad de partida, seguida de una permutación del resto de las ciudades. Para pasar de un estado a un sucesor, se añade una ciudad no visitada aún, y el coste correspondiente es el coste de ir de la última ciudad visitada en el estado actual, a la ciudad nueva. En el caso de que el nuevo estado sea un objetivo, hay que añadir al coste de la regla el coste para ir desde la ciudad añadida hasta la ciudad de partida. La Figura 8.8 muestra

el grafo de conexiones para una instancia del problema TSP con 6 ciudades, siendo A la ciudad de partida, y la Figura 8.9 una parte del espacio de búsqueda para esta instancia. El coste del objetivo ($A C B E F D$) es $12 + 7 + 25 + 17 + 39 + 15 = 115$.

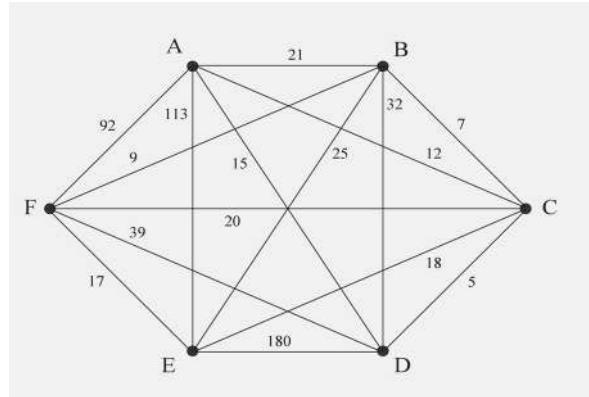


Figura 8.8: Grafo de conexiones para una instancia del problema TSP.

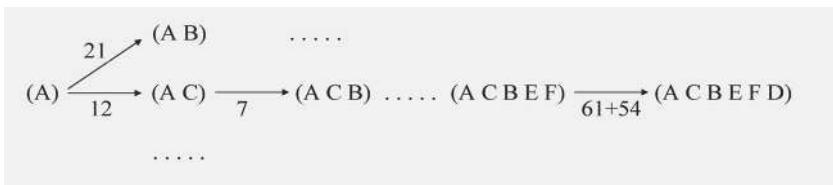


Figura 8.9: Espacio de búsqueda para la instancia del problema TSP de la Figura 8.8.

El espacio de búsqueda de la Figura 8.9 es completo y tiene estructura de árbol. Sin embargo, se trata de un espacio redundante, ya que hay varios nodos que representan realmente el mismo subproblema. Por ejemplo, los estados $(A C B E)$ y $(A B C E)$ representan los dos el problema de ir desde la ciudad E a la ciudad A pasando una sola vez por las ciudades F y G. De este modo, se puede hacer una segunda interpretación de los estados como situaciones en las que el viajante ha pasado a través de un subconjunto de ciudades y que se encuentra en una determinada ciudad. Con esta interpretación, el espacio de búsqueda no presenta redundancias y por lo tanto es más reducido, pero no tiene estructura de árbol. La Figura 8.10 muestra el espacio de búsqueda para el problema de la Figura 8.8 de acuerdo con esta segunda interpretación. El estado $(A B C E)$ representa que el viajante ha partido de la ciudad A, que ha pasado a través de las ciudades B y C, en un orden no determinado, y que se encuentra en la ciudad E. Como se puede observar, el número de estados objetivo se reduce a 1, frente a $(N - 1)!$ estados de la primera versión. También se puede observar que el número de estados intermedios es mucho menor. Esta segunda versión del espacio de búsqueda enfatiza que el significado real de un estado es un subproblema sin resolver. Este subproblema es una versión reducida del problema

original (el que representa el estado *initial*) y cada uno de los caminos desde el inicial al nodo representa una forma de reducir el problema inicial al que representa el nodo. El cometido de los algoritmos de búsqueda es encontrar el mejor de estos caminos.

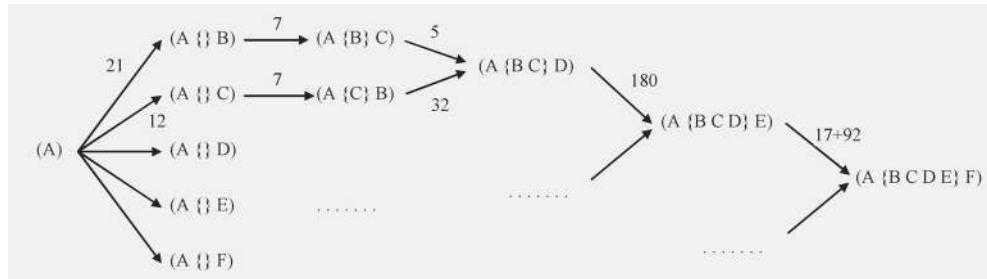


Figura 8.10: Espacio de búsqueda para la instancia del problema TSP de la Figura 8.8 con la segunda versión.

■

8.3. El problema del 8-puzzle. Al igual que los dos problemas anteriores, los problemas de solución de puzzles son ampliamente utilizados en los textos y trabajos de investigación sobre algoritmos de búsqueda, porque tienen una formulación simple y a la vez son complejos de resolver.

El problema del 8-puzzle consiste en redistribuir un conjunto de 8 fichas colocadas sobre un tablero de dimensiones 3×3 , para obtener una determinada configuración objetivo. La redistribución solamente se puede hacer mediante una secuencia de desplazamientos elementales de fichas, en cada uno de los cuales una ficha se puede mover a la posición vacía, si esta posición es adyacente ortogonalmente a la posición actual de la ficha. La Figura 8.11 muestra la situación inicial, para una instancia de este problema, los sucesores de este estado, así como el estado objetivo que se pretende alcanzar.

La modelización de los estados y de los operadores en este caso es bastante simple, así como la caracterización del objetivo que, como es único, se puede representar de forma explícita. El espacio de búsqueda es un grafo que contiene ciclos; de hecho los operadores son reversibles, con lo que parece un problema adecuado para aplicar un método de búsqueda bidireccional. En cualquier caso hay que utilizar un algoritmo de búsqueda en grafos, ya que claramente se pueden encontrar varios caminos desde el nodo *initial* a un estado intermedio con costes diferentes.

Este problema tiene algunas características que lo diferencian del problema de las N -reinas. Por ejemplo, en el problema de las N -reinas, una vez obtenida una solución, el camino desde el estado *initial* hasta la solución está implícito en la propia solución; sin embargo en el 8-puzzle eso no es así. De hecho en el 8-puzzle se conoce a priori el estado solución y el objetivo de la búsqueda es encontrar el camino desde el *initial*

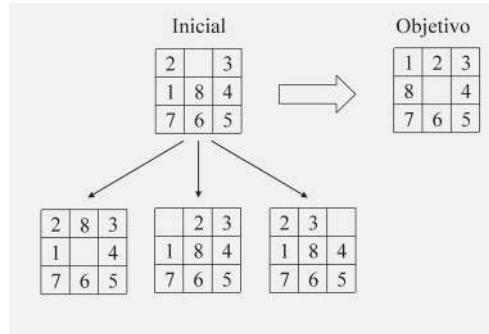


Figura 8.11: Un estado inicial, sus sucesores y el objetivo para una instancia del problema del 8-puzzle.

hasta el objetivo, no el propio objetivo. Mientras que en el problema de las N -reinas el objetivo de la búsqueda es encontrar un estado solución, de los muchos posibles.

El problema del cálculo de planes de actuación de robots, y en general los problemas de planificación de actuaciones que se estudian en el capítulo 13, tienen características similares al problema del 8-puzzle. Otros problemas, como el problema de las N -reinas, pertenecen a la tipología de problemas de satisfacción de restricciones, los cuales serán estudiados en el capítulo 10.

■

8.4. Problemas de asignación de recursos a tareas. La organización de tareas que compiten por el uso de los mismos recursos ofrece una amplia variedad de problemas complejos, que se conocen como problemas de *scheduling*, y que tienen una formulación natural como problemas de búsqueda. En esta sección se considera la siguiente versión, denominada “secuenciamiento de tareas con cabezas y prioridades sobre una máquina”, que se puede enunciar como sigue: se tiene un conjunto de N tareas $\{t_1, \dots, t_N\}$ que deben ser realizadas todas ellas sobre una misma máquina m . Cada tarea i , $1 \leq i \leq N$, requiere el uso exclusivo e ininterrumpido de la máquina m durante un periodo de d_i unidades de tiempo, tiene un tiempo mínimo de inicio, o cabeza, $r_i \geq 0$ a partir del cual puede comenzar a ejecutarse y una prioridad p_i . El objetivo es calcular un orden de ejecución de las tareas $(t_{k_1}, \dots, t_{k_N})$ que minimice la suma de los tiempos de finalización ponderados por las prioridades, definida como

$$\sum_{i=1, \dots, N} T(k_i)p_{k_i}, \quad (8.9)$$

donde $T(k_1) = r_{k_1} + d_{k_1}$, y $T(k_i) = \max(T(k_{(i-1)}), r_{k_i}) + d_{k_i}$, $1 < i \leq N$. La Tabla 8.2 muestra los datos de una instancia del problema con 4 tareas.

En este problema el diseño del espacio de búsqueda ofrece varias posibilidades. En principio se puede considerar cada estado como una situación en la que algunas tareas

están planificadas, mientras que las restantes no lo están. Estas situaciones se pueden representar mediante subsecuencias o subconjuntos ordenados de tareas que expresan el orden de ejecución de esas tareas. El estado *inicial* será el conjunto vacío y los objetivos serán ordenaciones de todas las tareas. En consecuencia la aplicación de un operador consistirá en añadir una tarea, no incluida en el estado, al subconjunto de tareas del estado, para lo cual hay varias opciones. Una posibilidad, para calcular los sucesores de un estado, es considerar todas las tareas no incluidas en el estado y con cada una de ellas generar un sucesor añadiendo esa tarea al final de la secuencia de tareas del estado. Con esta opción, para la instancia de la Tabla 8.2, se obtendría el espacio de búsqueda que se muestra (parcialmente) en la Figura 8.12.

Tarea	Cabeza	Duración	Prioridad
t_1	5	3	2
t_2	6	4	3
t_3	12	7	2
t_4	14	5	3

Tabla 8.2: Instancia del problema de secuenciamiento de tareas con cabezas y prioridades sobre una máquina.

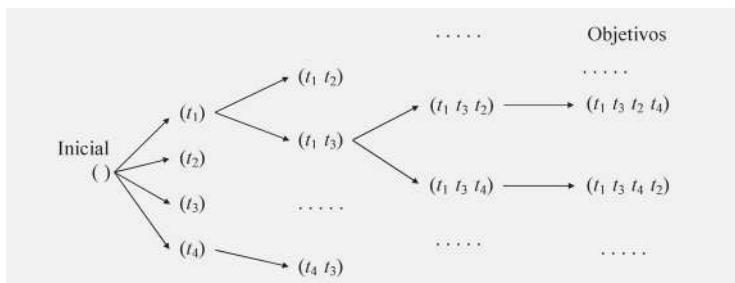


Figura 8.12: Árbol de búsqueda para el problema de la Tabla 8.2.

Para definir el coste de los operadores se debe tener en cuenta cómo varía la cantidad a optimizar, que viene dada por la expresión 8.9, al pasar de un estado al siguiente. Por ejemplo para el estado $(t_1 \ t_3)$ el coste total acumulado se calcula del siguiente modo, teniendo en cuenta que las dos tareas comienzan en el instante definido por su cabeza, 5 y 12 respectivamente, que sus duraciones son 3 y 7 respectivamente, y que las dos tienen prioridad igual a 2

$$(5 + 3) \times 2 + (12 + 7) \times 2 = 54$$

Análogamente el coste acumulado en el estado $(t_1 \ t_3 \ t_2)$ se calcula como

$$(5 + 3) \times 2 + (12 + 7) \times 2 + (19 + 4) \times 3 = 123$$

Con lo que el coste del operador que lleva del estado $(t_1 \ t_3)$ al estado $(t_1 \ t_3 \ t_2)$ es $123 - 54 = 69$.

El espacio de búsqueda de la Figura 8.12 es completo ya que incluye como nodos solución a las 24 soluciones posibles, definidas por cada una de las permutaciones de las 4 tareas del problema. Sin embargo, se puede observar que no es preciso un espacio de búsqueda que contenga a todas las soluciones posibles. Por ejemplo, el estado objetivo $(t_1 \ t_3 \ t_2 \ t_4)$ define la solución que se muestra, en forma de gráfico de Gantt, en la Figura 8.13. Se puede observar que la cabeza de la tarea t_2 es 6, luego sería posible adelantar la ejecución de esta tarea para ocupar el intervalo de tiempo entre los instantes 8 y 12, con lo que se podría adelantar también la ejecución de la tarea t_4 y así se obtendría una solución mejor, la misma que viene representada por el estado $(t_1 \ t_2 \ t_3 \ t_4)$.

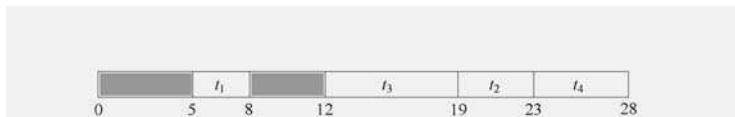


Figura 8.13: Solución correspondiente al estado objetivo $(t_1 \ t_3 \ t_2 \ t_4)$ del espacio de búsqueda de la Figura 8.12.

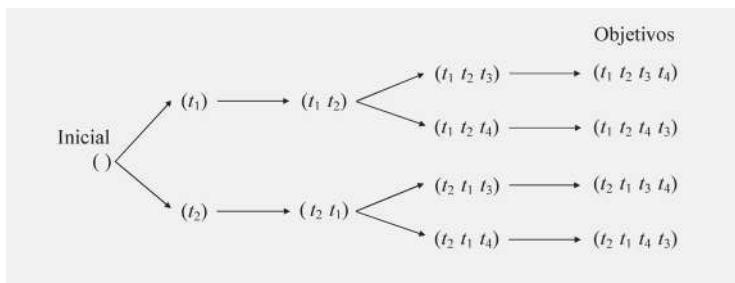


Figura 8.14: Segunda versión del espacio de búsqueda para el problema de la Tabla 8.2.

Para evitar soluciones “no interesantes” como la de la Figura 8.13, lo que se debe hacer es generar los sucesores de una forma más cuidadosa de modo que no se generen “huecos” que podrían ser ocupados por otras tareas que se planifican más tarde. Para ello, en este caso, basta considerar todas las tareas no incluidas en el estado y para cada una de ellas calcular el tiempo de finalización si se añade al estado para generar un sucesor. Se anota el mínimo de estos tiempos, T_{min} , y luego se filtran los sucesores de manera que se consideran solamente aquellos sucesores en los que la nueva tarea comienza en un tiempo estrictamente menor que T_{min} . El espacio de búsqueda generado de este modo para el problema de la Tabla 8.2 se muestra en la Figura 8.14. Como se puede observar es más reducido que el espacio de la Figura 8.14. Ahora hay 13 nodos frente a 65 nodos del espacio anterior. Este espacio de búsqueda es completo,

ya que contiene al menos una solución óptima, y al ser más reducido que el anterior, es de esperar que cualquier algoritmo de búsqueda sea más eficiente. Obviamente, puede darse el caso de que para una determinada instancia las dos versiones del espacio de búsqueda sean iguales.

■

8.7 Ejercicios propuestos

8.1. Dado que la búsqueda en espacios de estados se plantea como el cálculo de un camino de coste mínimo entre dos nodos de un grafo dirigido simple, se podría considerar como estrategia de búsqueda el conocido algoritmo de Dijkstra. Este algoritmo tiene una complejidad del orden de $O(N^2)$, donde N es el número de nodos del grafo, con lo que aparentemente se tendría una complejidad polinomial para resolver un problema NP-duro. Lo que se pide en este ejercicio es comentar esta posibilidad y, obviamente, resolver la paradoja anterior.

8.2. Estudiar los requerimientos computacionales de tiempo y de espacio de los algoritmos de búsqueda iterativa en anchura y búsqueda iterativa en profundidad.

8.3. Describir, en un lenguaje similar al empleado en los algoritmos descritos en este capítulo, un método de búsqueda bidireccional con dos algoritmos de búsqueda en anchura. Representar el espacio de búsqueda desarrollado por este algoritmo para encontrar una solución de algunas instancias del problema del 8-puzzle cuyas soluciones óptimas precisen un número entre 6 y 10 desplazamientos.

8.4. Modelar el espacio de búsqueda para resolver el conocido problema de los misioneros y los caníbales que se define de la siguiente forma. Tres misioneros y tres caníbales se encuentran en una orilla de un río con un bote que puede transportar a un máximo de dos personas. El objetivo es encontrar una secuencia mínima de desplazamientos del bote entre las dos orillas del río para llevar a todos los misioneros y a los caníbales a la otra orilla del río, de modo que nunca se encuentre en la misma orilla un grupo con más caníbales que misioneros.

8.5. Describir una versión del algoritmo de búsqueda primero en anchura en árboles, alternativo al que se muestra en el algoritmo 8.2, que compruebe si un estado q es objetivo en el momento en que este estado aparezca como sucesor del nodo que se expande. ¿Qué se puede decir de la admisibilidad de esta versión?

8.6. Considérese alguno de los métodos deductivos estudiados en el capítulo 2, por ejemplo la resolución SLD. ¿Existe alguna relación entre este método y los algoritmos estudiados en este capítulo? En caso afirmativo, ¿de cuál de ellos se trata?, ¿cómo son los estados y los operadores?, ¿tendría sentido la aplicación de una estrategia de control diferente?

8.7. Análogo al anterior, pero considerando los métodos de encadenamiento de reglas del capítulo 3. ¿Cómo son en este caso los estados y los operadores?

8.8. Resolver mediante un sistema de búsqueda en espacios de estados el conocido problema QAP (Quadratic Assignment Problem) que se define del siguiente modo. Se dispone de un conjunto de N unidades productivas que deben ser ubicadas en un conjunto de otras tantas posiciones (cada posición tiene capacidad para una sola unidad). Las unidades productivas requieren un intercambio de materiales entre ellas que viene cuantificado por los elementos de la matriz de flujo A , de forma que A_{ij} es el flujo que va desde la unidad i a la unidad j , $1 < i, j < N$. (puede ser $A_{ij} \neq A_{ji}$). Se tiene también una matriz de distancias entre las posiciones, D , tal que D_{kl} es la distancia desde la posición k a la posición l , $1 < k, l < N$ (también puede ser $D_{kl} \neq D_{lk}$). El coste de movimientos de mercancías desde la unidad i que está en la posición k hasta la unidad j que se encuentra en la posición l viene dado por

$$A_{ij} \times D_{kl}$$

El objetivo es conseguir una ubicación de las unidades que minimice el coste total definido como

$$\sum_{i=1}^N \sum_{j=1}^N A_{ij} \times D_{p(i)p(j)}$$

dónde $p(i)$ y $p(j)$ representan las posiciones de las unidades i y j respectivamente.

8.9. Resolver mediante búsqueda en espacios de estados el problema de cálculo del árbol de expansión mínimo con grado limitado, o LDMST (Limited Degree Minimum Spanning Tree). La formulación del problema es la siguiente. Dado un grafo G no dirigido y con costes positivos en los ejes, se trata de calcular un árbol de expansión mínimo en el que el grado (numero de ejes emergentes) máximo de los nodos es un valor $n \geq 2$. Es decir, se trata del problema MST clásico con una restricción adicional en el grado de los nodos. El problema MST es polinomial y se resuelve con los conocidos algoritmos de Prim o de Kruskal, sin embargo el LDMST es un problema NP-duro; en consecuencia lo razonable es intentar resolverlo mediante un algoritmo de búsqueda.

Describir una parte significativa del espacio de búsqueda para la instancia definida por el grafo de la Figura 8.8, suponiendo un grado máximo $n = 3$. ¿Se trata de un árbol o de un grafo?

Referencias

- FERNÁNDEZ, S.; GONZÁLEZ, J. y MIRA, J.: *Problemas resueltos de Inteligencia Artificial.* Addison Wesley, 1998.
- NILSSON, N. J.: *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.
- NILSSON, N. J.: *Principios de Inteligencia Artificial.* Díaz de Santos, Madrid, 1987.
- NILSSON, N. J.: *Artificial Intelligence. A New Synthesis.* Morgan Kaufmann, San Francisco, California, 1998.
- NILSSON, N. J.: *Inteligencia Artificial. Una Nueva Síntesis.* McGraw Hill, Madrid, 2001.
- PEARL, J.: *Heuristics. Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley, Massachusetts, 1984.
- RUSSELL, S. y NORVIG, P.: *Artificial Intelligence. A Modern Approach.* Prentice Hall, US, 2^a edición, 2003.

Capítulo 9

Técnicas basadas en búsquedas heurísticas

María Camino Rodríguez Vela y Ramiro Varela Arias

Universidad de Oviedo

9.1 Introducción

En el capítulo anterior se ha visto cómo las técnicas de búsqueda se pueden aplicar para la resolución de problemas. El procedimiento consiste simplemente en describir un espacio de búsqueda y luego elegir una estrategia para recorrer este espacio en busca de soluciones. El problema de utilizar una estrategia de búsqueda no informada, o búsqueda a ciegas, es que al realizar una búsqueda sistemática sin ningún tipo de información que le permita discernir las regiones más prometedoras de aquellas que lo son menos, es que el tiempo necesario para encontrar una solución puede llegar a ser prohibitivo. La alternativa es disponer de algún mecanismo que permita dirigir la búsqueda hacia las zonas más prometedoras, de modo que se pueda llegar a una solución sin necesidad de visitar tantos nodos como los que en general requiere una estrategia de búsqueda a ciegas. En el ámbito de la IA, a estos mecanismos se les denomina de forma genérica *heurísticos* o *heurísticas*. La palabra *heurístico* procede de la palabra griega *heuriskein*, que significa descubrir, al igual que la palabra *eureka* que pronunció Arquímedes en su baño justo después de descubrir el famoso principio que lleva su nombre.

Los *heurísticos* son criterios, reglas o métodos que ayudan a decidir cuál es la mejor alternativa entre varias posibles para alcanzar un determinado objetivo. Para ello, deben disponer de información, o mejor aun, de conocimiento sobre el problema que se intenta resolver. Este conocimiento se puede obtener a partir de cualquier pista, intuición o experiencia que se tenga sobre el dominio del problema. En el contexto de los sistemas de búsqueda, los *heurísticos* se suelen utilizar para decidir cuál de los nodos candidatos a ser expandidos es más prometedor, o bien en qué orden se deben aplicar las reglas a un nodo para generar sucesores, o incluso para decidir si un nodo representa una situación sin salida aun teniendo sucesores válidos.

De este modo, lo que se espera del uso de conocimiento en la búsqueda es que el número de nodos examinados para llegar a una solución se reduzca notablemente con respecto a una búsqueda sin conocimiento. Pero el uso de conocimiento introduce, como es lógico, un nuevo factor de coste en la búsqueda, ya que se deben evaluar las alternativas en función del conocimiento disponible. En consecuencia, hay que llegar a una situación de compromiso, como la que se muestra en la Figura 9.1, para explotar el conocimiento de forma que se optimice el coste total de la búsqueda. Si se explota de forma exhaustiva el conocimiento, se puede lograr una gran reducción del número de nodos a visitar para alcanzar una solución, pero el coste de gestión del conocimiento puede ser demasiado elevado, lo que incrementa el coste total de la búsqueda pudiendo éste llegar a ser excesivo, tal y como se muestra en la parte derecha de la gráfica de la Figura 9.1. En el caso contrario, si se utiliza poco, o ningún conocimiento, se puede tener un coste muy elevado debido a la aplicación de las reglas, como indica la parte derecha de la gráfica. El objetivo aquí será, por supuesto, situarse en la parte central de la gráfica, en donde se utiliza la cantidad de conocimiento apropiada para obtener el mínimo coste en el proceso de búsqueda. Lógicamente, este punto intermedio no siempre es fácil de localizar.

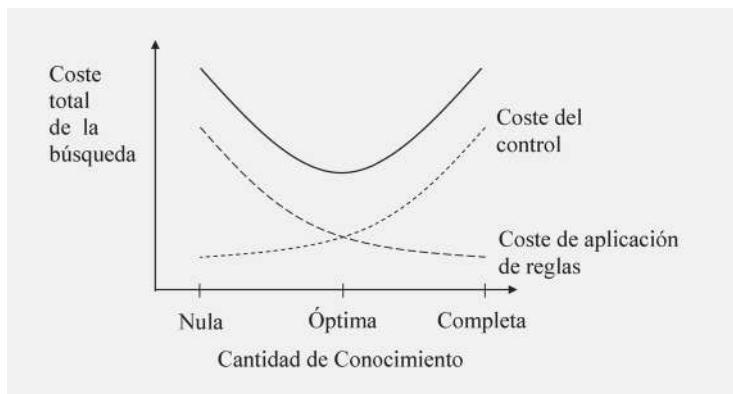


Figura 9.1: Representación del coste total de la búsqueda frente al grado de conocimiento utilizado.

Dado que los heurísticos se basan en el uso de conocimiento a veces impreciso o incierto, es natural que en muchas ocasiones fallen en la predicción de la mejor alternativa. Un buen heurístico es aquel capaz de tomar una buena decisión, no necesariamente la mejor, en un alto porcentaje de ocasiones. En consecuencia, el resultado de un buen heurístico es que el tiempo medio para llegar a una solución mejore notablemente, aunque en el peor de los casos el tiempo pueda ser igual que el de una búsqueda a ciegas.

El término heurístico se utiliza muchas veces en contraposición con el término algorítmico, para hacer referencia a procedimientos que no garantizan la calidad de la solución, ni en ocasiones la completitud. Sin embargo, también hay algoritmos exactos en los que la búsqueda se guía mediante heurísticos.

Para ilustrar cómo se puede utilizar el conocimiento en la búsqueda, y en particular para diferenciar lo que suele ser el conocimiento básico del problema que permite definir el espacio de búsqueda, del conocimiento más sofisticado que permite guiar la búsqueda en una buena dirección, se considera aquí el siguiente ejemplo. Se trata del conocido problema de la lechera, que con algunas diferencias de interpretación ha aparecido también en otros contextos.

El problema de la lechera se define de la siguiente forma: Se dispone de dos medidores de líquidos; uno de 5 litros y el otro de 7. Inicialmente ambos están vacíos y en cualquier momento pueden ser llenados o vaciados usando un depósito con suficiente capacidad. El líquido se trata, evidentemente, de leche y también puede ser traspasado de uno a otro medidor hasta que el primero se vacíe o hasta que el segundo se llene. El problema consiste en encontrar la secuencia de acciones que se deben seguir para que en el mayor de los medidores se disponga exactamente de 4 litros. Dado que el depósito tiene una cantidad de leche suficiente, lo relevante de los estados será la situación de los dos medidores. Así, un estado vendrá dado por el valor de un par de variables (x, y) con $0 \leq x \leq 7$ y $0 \leq y \leq 5$. El estado inicial será $(0, 0)$ y los estados objetivo serán de la forma $(4, y)$ con $0 \leq y \leq 5$. Teniendo en cuenta los movimientos posibles de leche entre los medidores y entre éstos y el depósito, los operadores pueden ser los siguientes; los 4 primeros se pueden aplicar de forma incondicional a cualquier estado, mientras que los 4 siguientes son aplicables sólo si se cumplen los prerequisitos que se indican en cada caso:

1. *Llenar el mayor:* $[(x, y) \rightarrow (7, y)]$
2. *Llenar el menor:* $[(x, y) \rightarrow (x, 5)]$
3. *Vaciar el mayor:* $[(x, y) \rightarrow (0, y)]$
4. *Vaciar el menor:* $[(x, y) \rightarrow (x, 0)]$
5. *Transferir del mayor al menor hasta que el primero se vacíe:*

Prerrequisitos: $(x + y < 5)$

$[(x, y) \rightarrow (0, x + y)]$

6. *Transferir del menor al mayor hasta que el primero se vacíe:*

Prerrequisitos: $(x + y < 7)$

$[(x, y) \rightarrow (x + y, 0)]$

7. *Transferir del menor al mayor hasta que se llene el mayor:*

Prerrequisitos: $(x + y > 7)$

$[(x, y) \rightarrow (7, x + y - 7)]$

8. *Transferir del mayor al menor hasta que se llene el menor:*

Prerrequisitos: $(x + y > 5)$

$[(x, y) \rightarrow (x + y - 5, 5)]$

Es claro que una aplicación sistemática de estas reglas debería encontrar la solución (si existe) a los problemas que la lechera pueda encontrar ya que, de hecho, estas son todas las acciones permitidas en este entorno.

No obstante, es seguro que no son contempladas todas ellas de la misma forma por una experimentada lechera al tratar de resolver el problema de medir 4 litros; sino que su experiencia le dictará algún modo de manejarlas que no será usando un algoritmo de búsqueda a ciegas. Además, es posible que la experta lechera sea incapaz de transmitirlas con precisión a otro interlocutor, como por ejemplo a un programador que intente diseñar un sistema de búsqueda para resolver el problema. El análisis del comportamiento de la lechera, con el fin de explicitar su conocimiento para poder incorporarlo en un heurístico, es una tarea de ingeniería del conocimiento que en este caso podría dar lugar a otro conjunto de reglas sobre cómo aplicar de forma ordenada las reglas anteriores. Este tipo de reglas sobre cómo utilizar las reglas se suele denominar metarreglas y, en este caso, (obtención de 4 litros en el mayor medidor) se puede sintetizar así:

1. *Siempre que el mayor esté vacío, llenarlo:*

Prerrequisitos: ($x = 0$)

[Aplicar la regla 1]

2. *Siempre que el pequeño esté lleno, vaciarlo:*

Prerrequisitos: ($y = 5$)

[Aplicar la regla 4]

3. *En otros casos, transferir toda la leche que se pueda del mayor al menor:*

Prerrequisitos: $\neg((x = 0) \circ (y = 5))$

[Si es posible aplicar la regla 8 si no aplicar la regla 5]

El problema consiste en saber si siguiendo estas metarreglas se obtiene una solución, si existe alguna; es decir, si constituyen un sistema completo. En este caso se puede probar que el sistema de metarreglas es completo; pero otras veces, por supuesto, esta prueba no es trivial.

En este capítulo se estudiará una variedad de algoritmos de búsqueda, muchos de los cuales son generalizaciones de los algoritmos de búsqueda a ciegas del capítulo anterior, que permiten incorporar conocimiento sobre el problema. Se analizarán las propiedades de estos algoritmos y se verá cómo aplicarlos a problemas concretos. Para ello se han elegido problemas formales, fáciles de enunciar y suficientemente complejos de resolver de modo que se pueda apreciar con claridad la eficiencia que proporciona el uso de conocimiento en la búsqueda de soluciones.

9.2 Búsqueda primero el mejor

El algoritmo de búsqueda primero el mejor, o BF (Best First), es una especialización del algoritmo general de búsqueda en grafos en el que se utiliza una función de evaluación de los nodos, f , tal que para cada nodo n , $f(n)$ da un valor numérico que indica una medida de lo prometedor que es el nodo para ser expandido. Esta función se utiliza para ordenar la lista *ABIERTA*, de modo que aquellos nodos más prometedores estarán al principio. La medida de lo prometedor que resulta un nodo para su expansión se puede estimar de varias formas. Por ejemplo, se puede tratar de medir la dificultad de resolver el subproblema que plantea el nodo, o bien estimar la calidad de las soluciones que se pueden obtener a partir de ese nodo. En cualquier caso, al definir la función f para un nodo n hay que tener en cuenta la descripción del propio estado n , toda la información acumulada durante la búsqueda, y lo que suele ser más importante, cualquier tipo de información, pista o conocimiento que se tenga sobre el dominio del problema. Por esa razón, f se suele denominar *función heurística de evaluación*.

Por ejemplo, en el problema de las N -reinas se puede definir la función f teniendo en cuenta el número de reinas que faltan por colocar y el número de posiciones no “atacadas” por las reinas ya colocadas. Este último valor da una medida de lo difícil que resultará resolver el subproblema planteado por el nodo, ya que cuanto mayor sea el número de posiciones no atacadas, parece razonable estimar que más fácil será resolver el problema, o mayor será la probabilidad de que a partir de este estado se pueda llegar a una solución. En el problema del 8-puzzle, se puede estimar lo prometedor que es un estado en función de lo que se parece al estado objetivo, por ejemplo en función de lo próximas que están las fichas a la posición que tienen que ocupar en el objetivo. Es evidente que en los dos casos, la elección del nodo más prometedor en cada iteración no siempre va a llevar de forma directa a la mejor solución, pero como se verá a lo largo de este capítulo, este tipo de criterios heurísticos permiten, en general, llegar a buenas soluciones expandiendo un número de estados mucho menor que si las elecciones se hacen de forma puramente aleatoria.

9.3 El algoritmo A*

El algoritmo A* es a su vez una especialización del algoritmo BF en el que la función de evaluación f se define de una forma particular. Antes de ver esta definición de f se introducen algunos conceptos y notación. Sea n un nodo cualquiera del espacio de búsqueda

Definición 9.1. $g^*(n)$ es el coste del camino más corto, o de menor coste, desde el inicial a n .

Definición 9.2. $h^*(n)$ es el coste del camino más corto desde n al objetivo más cercano a n .

Definición 9.3. $f^*(n) = g^*(n) + h^*(n)$. Es decir, $f^*(n)$ es el coste del camino más corto desde el inicial a los objetivos condicionado a pasar por el nodo n .

Definición 9.4. $C^* = f^*(\text{initial}) = h^*(\text{initial})$. Es decir, C^* es el coste de la solución óptima.

Es evidente que si se pudiese determinar, con un método eficiente, los valores de las funciones g^* y h^* para todos los nodos, se dispondría de un algoritmo de búsqueda muy eficiente que iría de forma directa a la solución óptima. La idea sería simplemente expandir siempre el nodo de menor f^* , y en el caso de empate elegir un sucesor del último nodo expandido. De esta forma solamente se expandirían nodos de un camino óptimo. Para todos estos nodos se cumple que $f^*(n) = C^*$, mientras que para aquellos nodos que no están en un camino óptimo ocurre que $f^*(n) > C^*$. Luego f^* es un discriminante perfecto. Obviamente, para problemas complejos, los valores de estas funciones no se pueden calcular en un tiempo razonable, con lo cual el método de búsqueda anterior no es factible. Lo que sí se puede hacer es trabajar con aproximaciones de las funciones g^* y h^* . Esto es precisamente lo que se hace en el algoritmo A*, en el que se utilizan las siguientes aproximaciones.

Definición 9.5. $g(n)$ es el coste del mejor camino desde el inicial a n obtenido hasta el momento durante la búsqueda.

Definición 9.6. $h(n)$ es una estimación positiva del valor de $h^*(n)$, tal que $h(n) = 0$, si n es un objetivo.

De estas dos definiciones resulta obvio que $g(n) \geq g^*(n)$ y $h(n) \geq 0$. Por último se define la función f del algoritmo BF, para obtener el algoritmo A*, como

Definición 9.7. $f(n) = g(n) + h(n)$.

De este modo f es una estimación de f^* y se utiliza para ordenar *ABIERTA* de modo que siempre estará al principio de esta lista el nodo con menor valor de f .

La función h se suele denominar *función heurística* o simplemente *heurístico*. La definición de esta función es un problema no trivial, y para ello habrá que tener en cuenta todo tipo de información obtenida por métodos más o menos formales, como cálculos estadísticos, o cualquier tipo de intuiciones o conocimientos que suelen tener los expertos en la resolución del problema. Como se verá más adelante, las propiedades de esta función condicionan totalmente el comportamiento del algoritmo A*.

9.3.1 Descripción del algoritmo A*

El algoritmo A* es el mismo que el AGBG descrito en el Algoritmo 8.6 del capítulo 8 con algunos cambios debidos a la forma de definir la función de evaluación f . En la versión que se muestra en el Algoritmo 9.1, se hace una ligera modificación del contenido de la *TABLA_A* para registrar, para cada nodo n encontrado, el valor de $g(n)$ que es lo mismo que *Coste(initial, n)* del algoritmo general, y el valor de $h(n)$. En esta descripción se supone que el valor de $h(n)$, una vez encontrado el nodo n , no cambia a lo largo del tiempo. Como la evaluación de h tiene en general un coste no despreciable, el valor de $h(n)$ se calcula una sola vez para cada nodo n y se almacena este valor en la *TABLA_A* para su uso posterior.

Algoritmo 9.1 Algoritmo A*.

```

ABIERTA = (inicial);
mientras NoVacia(ABIERTA) hacer
    n = ExtraePrimero(ABIERTA);
    si EsObjetivo(n) entonces
        devolver Camino(inicial, n) y para;
    fin si
    S = Sucesores(n);
    Añade S a la entrada de n en la TABLA_A;
    para cada q de S hacer
        si ( $q \in TABLA_A$ ) entonces
            Rectificar(q, n, Coste(n, q));
            Ordenar(ABIERTA); {si es preciso}
        si no
            pone q en la TABLA_A con
                Anterior(q) = n,
                 $g(q) = g(n) + Coste(n, q)$ ,
                 $h(q) = Heuristico(q)$ ;
            ABIERTA = Mezclar(q, ABIERTA);
        fin si
    fin para
fin mientras
devolver "no solución";

```

Algoritmo 9.2 Funciones de rectificación de nodos ya expandidos.

```

Función Rectificar (n, p, costepn);
si ( $g(p) + costepn < g(n)$ ) entonces
    Modifica la entrada del nodo n en la TABLA_A con  $g(n) = g(p) + costepn$ ;  $Anterior(n) = p$ ;
    RectificarLista(n);
fin si

Función RectificarLista (n);
LISTA = Sucesores(n); /* Registrados en la TABLA_A */
para cada q de LISTA hacer
    Rectificar(q, n, Coste(n, q));
fin para

```

9.3.2 Propiedades formales

En esta sección se estudian las propiedades formales del algoritmo A* que, como ya se ha adelantado, dependen de la definición de la función heurística h . Concretamente, se considerarán las propiedades de terminación, completitud, admisibilidad y eficiencia. El estudio formal del algoritmo A* se puede ver con más nivel de detalle, que el que se presenta aquí, en los textos de N. J. Nilsson [Nilsson, 1971, 1987, 1998, 2001] y J. Pearl [Pearl, 1984].

Para estudiar estas propiedades se introduce la siguiente notación. En lo sucesivo, s denotará el estado *inicial* y Γ el conjunto de nodos objetivo. $\Gamma^* \subseteq \Gamma$ es el conjunto de objetivos óptimos. $P_{n-n'}$ denotará un camino simple desde el nodo n al nodo n' , y $P_{n-n'}^*$ un camino óptimo desde n a n' .

Para describir una situación durante el proceso de búsqueda se hará referencia a los conjuntos de nodos encontrados, nodos expandidos y nodos aún no encontrados respectivamente. Los nodos encontrados son aquellos que ya tienen una entrada en la *TABLA_A*, los nodos expandidos son aquellos que fueron seleccionados para su expansión y que por lo tanto ya tienen registrados sus sucesores en el campo correspondiente de la *TABLA_A*. Y los nodos no encontrados son todos aquellos que aún no han aparecido durante la búsqueda. Así, justo antes de la expansión de un nodo, se da siempre la siguiente relación

$$\text{NodosEnABIERTA} \cup \text{NodosExpandidos} = \text{NodosEncontrados},$$

donde la unión es disjunta, es decir, *ABIERTA* es una frontera entre los nodos ya expandidos y aquellos no encontrados todavía durante la búsqueda. En consecuencia, cualquier camino $P_{n-n'}$, con n ya expandido y n' no expandido aún, tiene necesariamente al menos un nodo en *ABIERTA*.

9.3.2.1 Terminación y completitud

El algoritmo BF, descrito en la sección 9.2, es completo en grafos finitos, ya que en el peor de los casos terminaría después de expandir todos los nodos. Esto es evidente porque el algoritmo expande un nodo en cada iteración y ningún nodo se expande más de una vez. Sin embargo, el algoritmo BF puede no terminar en grafos infinitos, aunque sean localmente finitos¹, debido a que la definición de la función f no garantiza que el algoritmo no se vaya por una rama infinita del espacio de búsqueda. Sin embargo, esto no puede ocurrir en el algoritmo A*, debido a la definición de $f = g + h$. La componente h siempre es positiva, y la componente g crece de forma no acotada a lo largo de cualquier camino con un número no finito de nodos, por lo que si la búsqueda se fuese a través de un camino infinito, en algún momento el valor de f sería suficientemente grande para que el siguiente nodo a través de ese camino no fuese el más prometedor, y por lo tanto se consideraría la búsqueda a través de otros caminos. Por lo tanto se puede enunciar el siguiente resultado:

Teorema 9.1. *A* es completo en grafos localmente finitos.*

9.3.2.2 Admisibilidad del algoritmo A*

En esta sección se estudia en qué condiciones el algoritmo A* es admisible, es decir que encuentra la solución óptima. Como se verá, para que esto ocurra la función heurística h debe ser una estimación optimista del valor de $h^*(n)$ para todos los nodos. Esto motiva la siguiente definición

Definición 9.8. *Una función heurística h es admisible si $h(n) \leq h^*(n) \forall n$.*

A partir de aquí se supone que la función h es admisible y se probará que el algoritmo A* que la utiliza es admisible. Como paso previo se prueba el siguiente resultado.

¹Un grafo es localmente finito si cada nodo tiene un número finito de sucesores.

Lema 9.1. Antes de que el algoritmo A* termine, existe un nodo n' en ABIERTA tal que $f(n') \leq C^*$.

Demuestra. Considérese un camino $P_{s-\gamma}^*$, siendo $\gamma \in \Gamma^*$, es decir, un objetivo óptimo. Al ser γ un nodo no expandido, es evidente que existe al menos un nodo de $P_{s-\gamma}^*$ en ABIERTA. Si hay varios nodos de $P_{s-\gamma}^*$ en ABIERTA, sea n el que está más próximo al nodo inicial s , es decir n es aquel para el cual todos sus antecesores en $P_{s-\gamma}^*$ ya han sido expandidos. Al haber sido expandidos todos los nodos a través de un camino óptimo desde s a n tendremos que $g(n) = g^*(n)$. Además al ser h admisible y n un nodo de un camino óptimo

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

Luego $f(n) \leq C^*$. □

A partir de este resultado se puede probar el siguiente teorema:

Teorema 9.2. A* es admisible.

Demuestra. Ya se ha visto que A* es completo. Supongamos ahora que A* termina encontrando un objetivo $t \in \Gamma$ con $f(t) = g(t) > C^*$. En el momento de ser elegido para expansión se cumplía que $f(t) \leq f(n) \forall n \in \text{ABIERTA}$. Pero por el Lema 9.1 se sabe que antes de terminar siempre hay en ABIERTA un nodo n tal que $f(n) \leq C^*$. Luego en ese momento no es posible que se haya elegido el nodo t . Se tiene así una contradicción con la suposición de que $f(t) > C^*$, por lo tanto dado que $t \in \Gamma$ se tiene que $f(t) = C^*$. □

Como corolario de lo anterior se tiene que los algoritmos de primero en anchura y de coste uniforme son admisibles, ya que estos dos algoritmos son casos particulares del algoritmo A* con $h(n) = 0 \forall n$. En consecuencia, cabe preguntarse si estos algoritmos resultan ser iguales o no a otras versiones de A* con cualquier otro heurístico admisible. La respuesta es claramente que no son iguales, tal y como se probará en las dos secciones siguientes, en las que se verá en qué condiciones un heurístico se puede considerar más eficiente que otro, siendo ambos admisibles; y en qué condiciones se puede eliminar la operación de rectificación de nodos ya expandidos, y en consecuencia la necesidad de registrar el grafo expandido hasta el momento.

9.3.2.3 Comparación de heurísticos admisibles

En esta sección se estudia cómo influye el heurístico en la eficiencia del algoritmo A*. Se supone que los heurísticos son admisibles y se tratará de probar en qué condiciones unos heurísticos son más eficientes que otros. Para ello, se supone también que el coste de evaluación de todas las funciones heurísticas es similar. Así, el coste, en tiempo de ejecución, de las distintas versiones del algoritmo A*, cuya única diferencia es el heurístico, será proporcional al número de nodos expandidos para llegar a una solución.

Considérese la situación que se muestra en la Figura 9.2 en la que hay dos heurísticos admisibles h_1 y h_2 , tales que $h_2(n) > h_1(n)$ para todo nodo n no final. La intuición

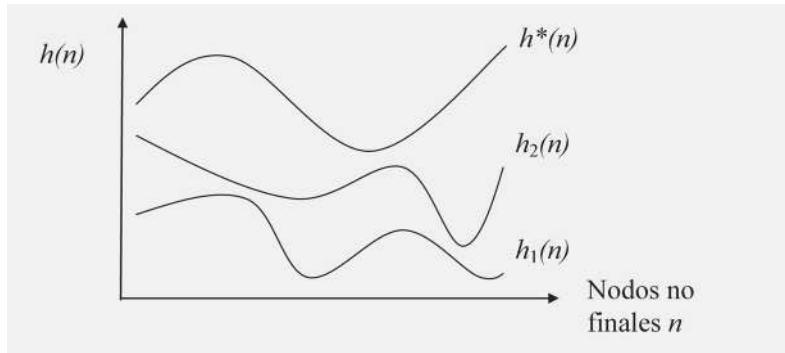


Figura 9.2: Dos heurísticos admisibles con distinto grado de información.

sugiere que h_2 contiene más información que h_1 ya que está más próximo al heurístico h^* que es el que contiene la información completa. Consecuentemente, el algoritmo A* debería ser más eficiente con h_2 que con h_1 . Lo que se puede probar en este caso es que todo nodo expandido con el heurístico h_2 será expandido también con h_1 , lo que significa que en el peor de los casos el número de nodos expandidos es el mismo. Sin embargo, si existe algún nodo no final n tal que $h_2(n) \leq h_1(n)$ este resultado formal ya no es cierto. Para probar que todo nodo expandido por h_2 también lo va a ser con h_1 el método que se sigue es, en primer lugar, establecer condiciones necesaria y suficiente de expansión. A continuación, se probará que si un nodo n cumple la condición necesaria de expansión con h_2 , entonces cumple la condición suficiente con h_1 . Para formalizar un poco más las ideas anteriores, se introducen las definiciones siguientes:

Definición 9.9. Un heurístico h_2 está más informado que otro heurístico h_1 si los dos son admisibles y $h_2(n) > h_1(n)$ para todo nodo n no final. Se dirá también que el algoritmo A_2^* que utiliza el heurístico h_2 está más informado que el algoritmo A_1^* que utiliza el heurístico h_1 .

Definición 9.10. Un algoritmo A_2^* domina a otro algoritmo A_1^* si los dos son admisibles y todo nodo expandido por A_2^* es también expandido por A_1^* .

Ahora se introducen las condiciones necesaria y suficiente de expansión para heurísticos admisibles.

Teorema 9.3. Una condición necesaria de expansión de un nodo n es que $f(n) \leq C^*$.

*Demuestra*ción. Es inmediata a partir del Lema 9.1, ya que si antes de terminar A* siempre hay en ABIERTA un nodo n tal que $f(n) \leq C^*$, todos los nodos expandidos deben cumplir esta condición. \square

Teorema 9.4. Una condición suficiente de expansión de un nodo n es que $f(n) < C^*$.

Demostración. También es inmediata, en este caso a partir del Teorema 9.2. Como el valor de f de un nodo solamente puede disminuir durante la búsqueda como consecuencia de una rectificación, cualquier nodo n con $f(n) < C^*$ debe ser expandido antes de que un objetivo t con $f(t) = C^*$ sea seleccionado para expansión. \square

Las condiciones dadas por los teoremas 9.3 y 9.4 anteriores son dinámicas, es decir dependen de la evolución del algoritmo A* ya que hacen referencia al valor de f que es $g + h$, y el valor de g es en realidad una variable del algoritmo que solamente está definida para los nodos encontrados. En consecuencia, del hecho de que un nodo n sea expandido con el heurístico h_2 , y que, por lo tanto, cumpla la condición necesaria de expansión dada por $g(n) + h_2(n) \leq C^*$, no se puede concluir que se cumple la condición suficiente de expansión con h_1 , es decir $g(n) + h_1(n) < C^*$, ya que no se puede suponer que con el heurístico h_1 el nodo n va a estar también en *ABIERTA* alguna vez y con el mismo valor de g que en el momento de su expansión con h_2 . No obstante, con ayuda de los dos teoremas anteriores se pueden probar otras dos condiciones de carácter estático, es decir que se pueden evaluar con independencia de la ejecución del algoritmo A*, únicamente a partir del espacio de búsqueda y del heurístico. La propiedad del espacio de búsqueda es la existencia de un camino C^* -acotado que se define del siguiente modo:

Definición 9.11. *Un camino P desde el inicial s a un nodo n es C -acotado, si para todo n' de P se cumple que $g_P(n') + h(n') \leq C$, y es estrictamente C -acotado si $g_P(n') + h(n') < C$. Siendo $g_P(n')$ el coste desde s a n' a través del camino P .*

Teorema 9.5. *Una condición suficiente para que el algoritmo A* expanda un nodo n es que existe un camino estrictamente C^* -acotado desde s a n .*

Demostración. Supongamos que existe un camino P desde s a n estrictamente C^* -acotado y que el algoritmo A* termina sin expandir n . Sea n' el nodo de P que está en *ABIERTA* al terminar A* para el que todos sus antecesores a través de P ya fueron expandidos, en este caso $g(n') \leq g_P(n')$. Luego, como P es estrictamente C^* -acotado

$$f(n') = g(n') + h(n') \leq g_P(n') + h(n') < C^*.$$

se tiene así una contradicción, ya que al terminar queda en *ABIERTA* el nodo n' con $f(n') < C^*$. \square

Teorema 9.6. *Una condición necesaria de expansión de un nodo n es que existe un camino C^* -acotado desde s a n .*

Demostración. Sea P el mejor camino desde s a n registrado en la *TABLA_A* en el momento de la expansión de n . Como todos los nodos n' de P ya han sido expandidos se tiene que $g(n') = g_P(n')$, y que $f(n') \leq C^*$, luego el camino P es C^* -acotado ya que para todo n' de P se cumple que $g_P(n') + h(n') = f(n') \leq C^*$. \square

Con la ayuda de estas dos condiciones estáticas ya se puede probar el resultado definitivo de dominación.

Teorema 9.7. *Si A_2^* está mas informado que A_1^* , entonces A_2^* domina a A_1^* .*

Demostración. Si n es expandido por A_2^* entonces existe un camino P desde s a n C^* -acotado con h_2 , es decir, para todo n' de P , $g_P(n') + h_2(n') \leq C^*$. Como $h_2(n') > h_1(n')$, se tiene que $g_P(n') + h_1(n') < C^*$. Luego el camino P es estrictamente C^* -acotado con h_1 , con lo cual n será expandido también por A_1^* . \square

El hecho de que tenga que cumplirse que $h_2(n) > h_1(n)$ para todo n no final, en la práctica, hace que este resultado sea de aplicación bastante limitada. Como se verá en los ejemplos, hay bastantes situaciones en las que se tienen dos heurísticos que cumplen esta relación para la mayoría de los estados no finales, pero hay unos pocos, normalmente estados muy próximos a los objetivos, para los que los dos heurísticos toman el mismo valor. Esto es lo que ocurre por ejemplo con los heurísticos h_1 y h_2 que se proponen para el problema del 8-puzzle en la sección 9.3.3.1; en general $h_2(n) > h_1(n)$, sin embargo, para los estados en los que solamente hay una ficha descolocada y que está a distancia 1 de su posición en el objetivo se da la igualdad $h_2(n) = h_1(n)$. En estas situaciones, lo que ocurre normalmente es que el heurístico h_2 se comporta mucho mejor que el heurístico h_1 , y en general esta mejora es tanto mayor cuanto mayor sea la diferencia entre los dos heurísticos en la mayoría de los estados intermedios.

9.3.2.4 Heurísticos Consistentes o Monótonos

En la sección anterior se ha visto la forma de comparar dos heurísticos admisibles a través del número de nodos expandidos. Sin embargo, un nodo expandido puede sufrir muchas rectificaciones posteriores, lo que evidentemente condiciona la eficiencia del algoritmo A^* . En esta sección se introducen dos nuevas propiedades de las funciones heurísticas: la consistencia y la monotonía. Como se verá, estas dos propiedades son equivalentes y, si el heurístico las cumple, aseguran que no es preciso rectificar el contenido de la TABLA_A para aquellos nodos que ya han sido expandidos.

Definición 9.12. *Un heurístico h es monótono si para todo par de nodos n y n' se cumple que*

$$h(n) \leq k(n, n') + h(n')$$

donde $k(n, n')$ representa el coste mínimo para ir de n a n' , y por lo tanto es infinito si no hay un camino desde n a n' .

Definición 9.13. *Un heurístico h es consistente si para todo par de nodos n y n' se cumple que*

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

donde $c(n, n')$ representa el coste de la regla que lleva de n a n' , y por lo tanto es infinito si no existe esta regla.

Es fácil ver que tanto h^* como el heurístico trivial $h(n) = 0$, para todo nodo n , cumplen las dos propiedades anteriores. Aunque en principio la monotonía parece menos restrictiva que la consistencia, en realidad se trata de propiedades equivalentes, como prueba el siguiente teorema.

Teorema 9.8. Monotonía y Consistencia son propiedades equivalentes.

Demostración. El hecho de que la monotonía implica consistencia es inmediato dado que $k(n, n') \leq c(n, n')$. Para probar que la consistencia implica monotonía hay que hacer un razonamiento por inducción en la profundidad de los sucesores de n , a través del camino más corto hasta n' . Sea este camino $P_{n-n'}^* = (n, n_1, \dots, n_i, n_{i+1}, \dots, n')$. Para el par de nodos (n, n_1) se cumple la condición de monotonía ya que $c(n, n_1) = k(n, n_1)$ por ser $P_{n-n'}^*$ óptimo. Supongamos que también se cumple esta propiedad para los nodos (n, n_i) , es decir que $h(n) \leq k(n, n_i) + h(n_i)$. Dado que $h(n_i) \leq c(n_i, n_{i+1}) + h(n_{i+1})$ por ser h consistente, y que $k(n_i, n_{i+1}) = c(n_i, n_{i+1})$ por ser $P_{n-n'}^*$ óptimo, se tiene

$$\begin{aligned} h(n) &\leq k(n, n_i) + h(n_i) \leq k(n, n_i) + c(n_i, n_{i+1}) + h(n_{i+1}) = \\ &= k(n, n_i) + k(n_i, n_{i+1}) + h(n_{i+1}) = k(n, n_{i+1}) + h(n_{i+1}). \end{aligned}$$

Puesto que el nodo n_{i+1} puede ser cualquiera de (n_1, \dots, n') se tiene que $h(n) \leq k(n, n') + h(n')$, luego h es monótono. \square

Una de las consecuencias de la monotonía o consistencia es la admisibilidad.

Teorema 9.9. Todo heurístico monótono es admisible.

Demostración. Sea n un nodo cualquiera y $\gamma \in \Gamma$ el objetivo más próximo a n , si h es monótono se tiene que

$$h(n) \leq k(n, \gamma) + h(\gamma)$$

como $h(\gamma) = 0$ y $k(n, \gamma) = h^*(n)$ se tiene que $h(n) \leq h^*(n)$. Luego h es admisible. \square

La implicación inversa no es cierta, es decir monotonía y admisibilidad no son propiedades equivalentes. Aunque en la práctica no es fácil encontrar heurísticos admisibles que no sean a la vez monótonos.

El uso de heurísticos monótonos tiene otras ventajas que ya se han adelantado. La primera de ellas es consecuencia del siguiente teorema.

Teorema 9.10. Si h es monótono y A^* elige el nodo n para expansión, entonces se cumple que $g(n) = g^*(n)$. Es decir que el camino encontrado hasta el momento desde el inicial s a n es óptimo.

Demostración. Supongamos que A^* elige n para expansión con $g(n) > g^*(n)$. Sea P_{s-n}^* un camino óptimo de s a n . Si n es el único nodo de P_{s-n}^* en ABIERTA, esta situación sería contradictoria, ya que estarían explorados todos los nodos a través de un camino óptimo con lo que tendría que ser $g(n) = g^*(n)$. Luego tiene que haber otro nodo de P_{s-n}^* en ABIERTA. Sea n' el nodo de P_{s-n}^* en ABIERTA más próximo a n ; para este nodo se cumple $g(n') = g^*(n')$, y por ser h monótono

$$f(n') = g^*(n') + h(n') \leq g^*(n') + k(n', n) + h(n)$$

como P_{s-n}^* es óptimo, $g^*(n') + k(n', n) = g^*(n)$ y dado que por hipótesis $g(n) > g^*(n)$ se tiene que

$$f(n') \leq g^*(n) + h(n) < g(n) + h(n) = f(n).$$

Es decir, se eligió un nodo n para expansión cuando en *ABIERTA* había un nodo n' con $f(n') < f(n)$, lo cual es una contradicción, ya que A^* siempre elige el nodo con menor f . Luego no puede ser $g(n) > g^*(n)$ y por lo tanto $g(n) = g^*(n)$. \square

De acuerdo con el Teorema 9.10, si el heurístico es monótono, no es necesario rectificar el camino desde el inicial para los nodos que ya fueron expandidos, con lo cual tampoco es necesario registrar el grafo de búsqueda desarrollado hasta el momento en el campo correspondiente a los sucesores de la *TABLA_A*, ya que los únicos nodos que pueden sufrir rectificaciones son los que se encuentran en *ABIERTA*, por lo que la operación *RectificarLista* nunca se aplica. Con lo que el algoritmo A^* es más eficiente. Además, como se probará a continuación, en el caso de que $h_2(n) \geq h_1(n)$ para todo n , es decir que se produzca la igualdad de los dos heurísticos para algunos nodos, si los heurísticos son monótonos, se puede acotar de alguna manera el número de nodos que pueden ser expandidos por h_2 y no por h_1 .

El teorema siguiente demuestra cómo quedan las condiciones, necesaria y suficiente, de expansión. Estas condiciones se simplifican con respecto a las condiciones de los heurísticos admisibles, ya que solamente requieren evaluar una condición simple sobre el nodo n y no sobre un camino desde s a n .

Teorema 9.11. *Si h es monótono, la condición necesaria de expansión de un nodo n es*

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

y la condición suficiente

$$g^*(n) + h(n) < C^*$$

Demostración. La condición necesaria se prueba de forma simple a partir de los teoremas 9.3 y 9.10. Si n se expande entonces $f(n) = g(n) + h(n) \leq C^*$, como $g(n) = g^*(n)$ en el momento de la expansión, se cumple que $g^*(n) + h(n) \leq C^*$.

Para probar la condición suficiente se utiliza el Teorema 9.5. La idea es demostrar que cualquier camino óptimo desde s a n es estrictamente C^* -acotado. Sea P_{s-n}^* un camino óptimo de s a n . Si $g^*(n) + h(n) < C^*$ y n' es el anterior a n en P_{s-n}^* , por ser P_{s-n}^* óptimo $g^*(n') + c(n', n) = g^*(n)$ y por ser h monótono $h(n') \leq c(n', n) + h(n)$, en consecuencia se tiene que $g^*(n') + h(n') < C^*$. Reiterando este razonamiento para el nodo anterior a n' y así sucesivamente se observa que todos los nodos de P_{s-n}^* cumplen esta condición, con lo que P_{s-n}^* es estrictamente C^* -acotado y por lo tanto el nodo n será expandido. \square

A partir de la condición suficiente de expansión, $g^*(n) + h(n) < C^*$, se puede ver que cuanto mejor informado sea el heurístico h , menor será el número de nodos que cumplen la condición y, por lo tanto, cabe esperar que el número de nodos expandidos sea menor. Las condiciones tienen también una implicación importante en el comportamiento de la regla de desempate entre los nodos con el mismo valor de f en *ABIERTA*. Si se consideran diferentes versiones de A^* , todas utilizando la misma función heurística, que se diferencian únicamente en la regla de desempate, los nodos que pueden ser expandidos por una versión y no por otra son aquellos que cumplen

$g^*(n) + h(n) = C^*$, que al estar en abierta tendrán $f(n) = C^*$. El algoritmo A* expandirá todos los nodos que cumplan $g^*(n) + h(n) < C^*$, alguno de los que cumplen $g^*(n) + h(n) = C^*$, y por supuesto ninguno de los que cumplen $g^*(n) + h(n) > C^*$. Se puede probar que cualquier algoritmo admisible que utilice la misma información h , tiene que expandir todos los nodos tales que $g^*(n) + h(n) < C^*$ para asegurar la optimidad de la solución, por esa razón el algoritmo A* cuando h es monótono domina ampliamente (véase la definición siguiente) a cualquier algoritmo admisible que utilice la misma información heurística. La prueba formal de este resultado se puede ver en [Dechther y Pearl, 1985] y [Pearl, 1984].

Además, estas dos condiciones permiten establecer lo que se llama una comparación no estricta entre dos algoritmos A_1^* y A_2^* que utilizan heurísticos monótonos h_1 y h_2 respectivamente, tales que $h_2(n) \geq h_1(n)$ para todo nodo n , es decir que no cumplen la desigualdad estricta para todos los nodos no finales. Para ello se introduce la siguiente definición.

Definición 9.14. *Un algoritmo A_2^* domina ampliamente a otro algoritmo A_1^* si todo nodo expandido por A_2^* es también expandido por A_1^* , excepto, quizás, algunos nodos para los cuales se cumple $h_2(n) = h_1(n) = C^* - g^*(n)$.*

Teorema 9.12. *Si $h_2(n) \geq h_1(n)$ para todo n y los dos son monótonos, entonces A_2^* domina ampliamente a A_1^* .*

Demuestra. Si un nodo n es expandido por A_2^* y no por A_1^* , entonces $g^*(n) + h_2(n) \leq C^*$ y $g^*(n) + h_1(n) \geq C^*$. Como $h_2(n) \geq h_1(n)$, se tiene que $h_2(n) = h_1(n) = C^* - g^*(n)$. \square

Este resultado permite tener un cierto grado de confianza en que el heurístico h_2 es más eficiente que el heurístico h_1 , ya que para que un nodo sea expandido por h_2 y no por h_1 , tienen que producirse dos igualdades, lo que en términos estadísticos puede considerarse un suceso poco probable, especialmente si los heurísticos toman valores reales.

9.3.3 Diseño de heurísticos simples

En esta sección se muestra cómo aplicar el algoritmo A* a una serie de problemas clásicos con distintas características. En principio se trata de seguir los pasos descritos en el capítulo anterior, es decir describir el espacio de búsqueda, y luego, lo que será más importante aquí, definir funciones heurísticas, a ser posible admisibles y monótonas, lo mejor informadas posible. En principio se definirán los heurísticos siguiendo criterios intuitivos en cada uno de los problemas. En la sección 9.3.5 se verá cómo este proceso se puede hacer, en muchos casos, de una forma más sistemática.

9.3.3.1 El problema del 8-puzzle

El problema del 8-puzzle ya ha sido formulado como un problema de búsqueda en espacios de estados en el capítulo anterior (ejercicio resuelto 3 de la sección 8.6). Aquí se considera el diseño de heurísticos. En este caso, para estimar el coste de resolver

el subproblema planteado por un estado n , se puede tener en cuenta la “diferencia” entre n y el objetivo que se desea alcanzar que es un estado conocido, como muestra la Figura 9.3. Una primera idea puede ser simplemente contar el número de fichas que en el estado n no están en la posición que les corresponde en el objetivo. Se tiene así el primer heurístico definido como

$$h_1(n) = \text{número de fichas que en el estado } n \text{ están fuera de su posición en el objetivo.}$$

Es claro que este heurístico es admisible, ya que cualquier ficha que esté descolocada debe ser desplazada al menos una vez para alcanzar la posición que le corresponde en el objetivo. También es claro que, salvo para estados que estén muy próximos al objetivo, este heurístico es una estimación muy optimista del coste que representa el estado n ; es decir que se trata de un heurístico muy poco informado. El valor de este heurístico para los tres sucesores del estado inicial del ejemplo de la Figura 9.3 es el siguiente:

$$h_1(n_1) = 3, \quad h_1(n_2) = 2, \quad h_1(n_3) = 4.$$

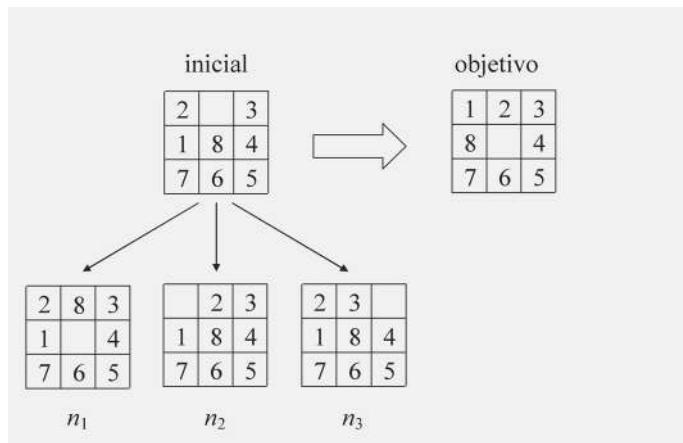


Figura 9.3: Un estado inicial, sus sucesores n_1 , n_2 y n_3 , y el objetivo para una instancia del problema del 8-puzzle.

Siguiendo la misma idea, se puede afinar un poco más y tener en cuenta las distancias de las fichas a su posición en el objetivo y definir otro heurístico como:

$$h_2(n) = \text{suma de las distancias ortogonales de cada ficha a su posición en el objetivo.}$$

También se trata de un heurístico admisible ya que cada ficha se debe mover al menos tantas veces como indica su distancia a la situación que le corresponde en el objetivo. Además, es una estimación menos optimista que la anterior, por lo que el heurístico h_2 resulta más informado que el heurístico h_1 . Concretamente se cumple que $h_2(n) \geq h_1(n)$ para todo nodo n . La igualdad se da para los estados cuyas fichas

descolocadas están a una distancia 1 de su posición en el objetivo. Para los tres nodos anteriores el heurístico h_2 toma los siguientes valores:

$$h_2(n_1) = 4, \quad h_2(n_2) = 2, \quad h_2(n_3) = 4.$$

Asimismo, es fácil probar que los dos heurísticos, h_1 y h_2 , son monótonos.

Para mostrar que se pueden definir heurísticos admisibles pero no monótonos, consideremos el siguiente, cuya definición es claramente poco natural:

$$h_3(n) = 2 \times \text{número de fichas que están a una distancia ortogonal 2 de su posición en el objetivo.}$$

Se trata de un heurístico admisible, menos informado que h_2 , y no monótono. Para probar esto último basta con encontrar un ejemplo en el que no se cumpla la desigualdad triangular.

9.3.3.2 El problema de las N -reinas

Este problema también ha sido formulado en el capítulo anterior y presenta algunas características por las que otros métodos de búsqueda resultan más eficientes que la búsqueda heurística en espacios de estados. Concretamente todas las soluciones tienen el mismo coste y están a la misma profundidad N ; por lo que el valor $h^*(n)$ para un estado en el que ya se han colocado k reinas, $0 \leq k \leq N$, es o bien $N - k$ si desde n se puede alcanzar una solución, o bien infinito si desde n no se alcanza ninguna solución, y obviamente el cálculo exacto no es trivial. Una primera consecuencia es que disponer de un heurístico admisible no tiene mucha trascendencia en este caso, ya que todas las soluciones son óptimas. Sin embargo, lo que se puede hacer es buscar un heurístico que guíe la búsqueda hacia estados desde los que se pueda alcanzar un objetivo con una alta probabilidad. Para ello, el heurístico puede tratar de estimar la probabilidad de que desde un estado se pueda alcanzar un objetivo; para lo cual la función heurística $h(n)$ debería retornar un valor pequeño si se estima que desde el estado n es probable que se alcance un objetivo, y un valor alto en caso contrario. Esta probabilidad se puede estimar de forma sencilla en función del número de casillas “no atacadas” que hay en el estado; sea este número $na(n)$. Así la probabilidad, $P_s(n)$, de que a partir de un estado n se pueda alcanzar una solución se puede estimar como

$$P_s(n) = \begin{cases} \frac{na(n)}{N(N-k)}, & \text{si } k < N \\ 1, & \text{si } k = N \end{cases} \quad (9.1)$$

Y definir, en principio el heurístico como

$$h(n) = 1 - P_s(n).$$

Además, para equilibrar los valores de g y h (obsérvese que $g(n) = k$) y favorecer así la expansión de los nodos más profundos entre aquellos que tienen la misma probabilidad de éxito, se puede ponderar el valor de h con el número de reinas que faltan por colocar, $N - k$, y así se tiene finalmente:

$$h(n) = (N - k)(1 - P_s(n)).$$

9.3.3.3 El problema del viajante de comercio

Este problema ya ha sido comentado también en el capítulo anterior (véase el ejercicio resuelto 2 de la sección 8.6). Para definir heurísticos, hay que estimar el coste de ir desde la última ciudad del estado a la ciudad de partida, pasando una y solo una vez por cada una de las ciudades no visitadas, es decir aquellas que no están incluidas en la secuencia de ciudades del estado. Para realizar estas estimaciones, de forma optimista, se puede razonar del siguiente modo. Si N es el número de ciudades del problema, y un estado n no objetivo contiene una secuencia de k ciudades, $1 \leq k \leq N$, para llegar a una solución desde n habrá que recorrer un número de $N - k + 1$ arcos. Luego se puede obtener una estimación optimista del coste de resolver n suponiendo que todos estos arcos tienen un coste igual al coste del arco mínimo de entre aquellos arcos no incluidos en la solución parcial, que representa el estado n , y que además no tocan a las ciudades intermedias. Obtenemos así el heurístico definido como:

$$h_1(n) = (N - k + 1) \times \text{coste del arco mínimo no incluido en } n \text{ y que no toca a las ciudades intermedias de } n.$$

Por ejemplo, el valor de este heurístico para los estados $n_1 = (\text{A})$, $n_2 = (\text{A B})$ y $n_3 = (\text{A C B})$ del ejemplo de la Figura 8.8 del capítulo 8 es el siguiente

$$h_1(n_1) = 6 \times 5 = 30, \quad h_1(n_2) = 5 \times 5 = 25, \quad h_1(n_3) = 4 \times 9 = 32.$$

Se trata claramente de un heurístico admisible, y se puede probar también que es monótono. Esto lo dejamos de momento como un ejercicio. Al igual que en el problema del 8-puzzle, se puede afinar un poco más en el razonamiento, y definir otro heurístico un poco mejor informado como:

$$h_2(n) = \text{Suma de los costes de los } (N - k + 1) \text{ arcos de coste mínimo no incluidos en } n \text{ y que no tocan a las ciudades intermedias de } n.$$

En este caso para los nodos anteriores los valores serán:

$$h_2(n_1) = 5 + 7 + 9 + 12 + 15 + 18 = 66, \quad h_2(n_2) = 5 + 7 + 9 + 12 + 15 = 48,$$

$$h_2(n_3) = 9 + 15 + 21 + 25 = 70.$$

También se puede definir un heurístico que intuitivamente aproximará mejor que los anteriores el valor de h^* como:

$$h_3(n) = (N - k + 1) \times \text{coste medio de los arcos no incluidos en } n \text{ y que no tocan a las ciudades intermedias de } n.$$

Aunque esta parezca, en general, una estimación muy razonable, claramente se trata de un heurístico no admisible. Se pueden utilizar otros criterios de razonamiento como el hecho de que para llegar de un estado n a una solución habrá que abandonar la última ciudad del estado, así como todas las no visitadas; o bien que habrá que llegar a todas las no visitadas y de nuevo a la ciudad de partida A. La definición de los heurísticos correspondientes se deja como ejercicio.

9.3.3.4 El problema del secuenciamiento de tareas con cabezas y prioridades

Este problema también ha sido formulado como un problema de búsqueda en espacios de estados en el capítulo anterior (ejercicio resuelto 4 de la sección 8.6). Para este problema, se puede hacer una estimación optimista del coste de resolver el subproblema planteado por un nodo n si se considera que todas las tareas que quedan por planificar tienen una prioridad igual a la prioridad mínima de todas ellas y también una duración igual a la duración mínima de todas ellas; y además que se planifican en el orden dado por sus cabezas. En el ejemplo del ejercicio resuelto 4, en concreto para los estados $n_1 = (t_1)$ y $n_2 = (t_1 \ t_2)$ de la Figura 8.14 del capítulo 8, el valor de este heurístico es:

$$h(n_1) = (12 + 16 + 20) \times 2 = 96, \quad h(n_2) = (17 + 22) \times 2 = 78.$$

El coste óptimo de cada nodo es claramente mayor que el valor del heurístico para el nodo, ya que en la solución óptima la suma de los tiempos de finalización de las tareas es mayor que la estimación que se hace en el heurístico y además cada uno de estos tiempos aparece multiplicado por un valor mayor o igual que la prioridad mínima de las tareas. Luego h es admisible. Sin embargo, se trata claramente de un heurístico poco informado, especialmente si hay diferencias muy grandes entre las duraciones de las tareas, o bien entre sus prioridades.

9.3.4 Relajación de las condiciones de optimalidad

En las secciones anteriores se ha visto en qué condiciones el algoritmo A* es capaz de encontrar soluciones óptimas. Dado que los problemas que se pretenden resolver con este algoritmo son muy complejos, en la práctica los heurísticos que se pueden diseñar, con un tiempo de evaluación aceptable, distan mucho de h^* para la mayoría de los nodos del espacio de búsqueda, por lo que la cantidad de nodos que es necesario expandir (al menos aquellos que cumplen la condición suficiente de expansión) es muy grande. Lo que se traduce, frecuentemente, en una necesidad de recursos computacionales (tiempo de búsqueda y espacio de almacenamiento) inaceptables. Tal y como apunta Pearl [Pearl, 1984, p. 86], la experiencia muestra que en muchos problemas A* pasa demasiado tiempo discriminando entre caminos cuyos costes no varían significativamente. En estos casos la admisibilidad es realmente un inconveniente más que una virtud, ya que fuerza al algoritmo a gastar un tiempo desproporcionado para seleccionar el mejor de entre un conjunto de candidatos prácticamente iguales y no permite que la búsqueda termine encontrando una solución aceptable, aunque no sea óptima.

Lo que se plantea ahora es la posibilidad de hacer que el algoritmo A* sea más eficiente, aunque para ello sea preciso perder la admisibilidad. Esto es lo que hay que hacer en la práctica con la mayoría de los problemas que se resuelven con técnicas de IA.

9.3.4.1 Ajuste de los pesos de g y h .

La estrategia de búsqueda de A^* está basada en la ecuación $f = g + h$. En ella, el sumando g trata de ajustar el algoritmo a una estrategia de búsqueda en anchura o de coste uniforme, lo que garantiza encontrar soluciones óptimas. Por el contrario, el sumando h trata de acercar la búsqueda directamente hacia la solución sin que, necesariamente, se expandan todos los nodos que disten igual del inicial. Una variante de este planteamiento sería efectuar una ponderación de ambas cantidades, es decir, considerar una función f_w de la forma:

$$f_w(n) = (1 - w)g(n) + wh(n), w \in [0, 1].$$

Claramente, $w = 0, 1/2$ y 1 corresponden a las estrategias de coste uniforme, A^* convencional y BF (con $f = h$) respectivamente. Si h es admisible, es fácil probar que el algoritmo es admisible en el intervalo $0 \leq w \leq 1/2$, pero se puede perder esta propiedad en el intervalo $1/2 \leq w \leq 1$, dependiendo de lo lejana que esté h de h^* .

El problema de este tipo de ponderación (que se llama estática, pues el peso w no varía a lo largo del proceso), es la gran dificultad que presenta encontrar, a priori, un valor adecuado para w . En la práctica se suele encontrar el mejor valor de forma experimental. Por ejemplo, algunos resultados experimentales con el problema del 15-puzzle [Pearl, 1984, p. 87] muestran que la mayor eficiencia se encuentra en el intervalo $1/2 \leq w \leq 1$. Sin embargo para el problema del 8-puzzle, lo mejor es $w = 1/2$, mientras que si se aumenta el valor, se obtiene un número mucho mayor de nodos expandidos.

9.3.4.2 Algoritmos ε -admisibles

Esta familia de algoritmos sacrifica la obtención de una solución óptima frente a la ventaja de obtener alguna mejora en el rendimiento del proceso, pero controlando el deterioro de la solución obtenida a través de un factor ε que representa la distancia máxima al coste óptimo.

Definición 9.15. *Un algoritmo de búsqueda es ε -admissible si siempre encuentra una solución con un coste que no excede el valor $(1 + \varepsilon)C^*$ (una solución de este tipo se denomina solución ε -óptima).*

En esta sección se verán las dos versiones más conocidas de este tipo de algoritmos: la ponderación dinámica y el algoritmo $A\varepsilon^*$.

Ponderación Dinámica. Esta estrategia se debe a Pohl [Pohl, 1973]. En lugar de fijar unos pesos estáticos, éstos se cambian de manera dinámica a lo largo del proceso de búsqueda. La idea es proporcionar al sumando h un mayor peso al principio de la búsqueda hasta que se esté en las proximidades de la solución, en donde se reduce este peso para asegurar una búsqueda en anchura que afine lo más posible la solución. La función de evaluación que se utiliza con este método es

$$f(n) = g(n) + h(n) + \varepsilon(1 - d(n)/N)h(n)$$

donde $d(n)$ es la profundidad del nodo n y N es una cota superior de la profundidad de la mejor solución. El número $\varepsilon > 0$ indica la desviación que estamos dispuestos a admitir.

No es difícil probar que si el heurístico es admisible, entonces el algoritmo es ε -admisible.

Teorema 9.13. *Si el heurístico h es admisible, el algoritmo de ponderación dinámica es ε -admisible.*

Demuestra. Hay que probar que si el problema tiene solución el algoritmo de ponderación dinámica devuelve una cuyo costo no excede $(1 + \varepsilon)C^*$. En primer lugar se debe probar que el algoritmo termina, lo cual es simple mediante un razonamiento similar al utilizado con el A* normal (Teorema 9.1).

Para ver que el algoritmo termina con una solución ε -óptima se probará que una condición necesaria de expansión de un nodo n es que $f(n) \leq (1 + \varepsilon)C^*$. El razonamiento es análogo al utilizado en el Lema 9.1. Si P es un camino óptimo del inicial a los objetivos, entonces antes de terminar siempre hay en *ABIERTA* un nodo n de P tal que todos sus antecesores a través de P ya fueron expandidos y por lo tanto $g(n) = g^*(n)$. Luego para el nodo n se tendrá que

$$\begin{aligned} f(n) &= g^*(n) + h(n) + \varepsilon[1 - d(n)/N]h(n) \\ &\leq g^*(n) + h^*(n) + \varepsilon[1 - d(n)/N]h^*(n) \\ &= C^* + \varepsilon[1 - d(n)/N]h^*(n) \\ &\leq C^* + \varepsilon h^*(n) \leq (1 + \varepsilon)C^*. \end{aligned}$$

Luego siempre hay en *ABIERTA* un nodo n con $f(n) \leq (1 + \varepsilon)C^*$ antes de terminar, por lo que todo nodo expandido debe tener un valor de f menor o igual que $(1 + \varepsilon)C^*$. En particular si se elige un objetivo t para expansión debe ocurrir que

$$f(t) = g(t) + h(t) = g(t) \leq (1 + \varepsilon)C^*$$

□

Algoritmo A ε^* . Esta versión es debida a J. Pearl y J. H. Kim [Pearl, 1984, p. 88]. La idea consiste en considerar dentro de *ABIERTA* una sublista, llamada *FOCAL*, formada por aquellos nodos cuyo valor de f no se separa más de un factor $(1 + \varepsilon)$ del valor mínimo de f en *ABIERTA*, es decir

$$FOCAL = \{n \in ABIERTA / f(n) \leq (1 + \varepsilon)\min(f(n'), n' \in ABIERTA)\}.$$

La estrategia de A ε^* es idéntica a la de A*, salvo que en A ε^* se desarrollan primero los nodos de la sublista *FOCAL* según los valores de un segundo heurístico h' , donde $h'(n)$ es una estimación del esfuerzo computacional que se necesitaría para llegar desde n a una solución.

La razón de este cambio es que los nodos de *FOCAL* tienen todos más o menos las mismas posibilidades de estar en un camino óptimo, según las estimaciones basadas

en f . Así pues, si se dispusiese de algún conocimiento adicional, que no pudiese ser representado en la función f , parece razonable utilizar este conocimiento para decidir cuál es el nodo más prometedor.

Obsérvese que como heurístico h' podría utilizarse el propio h , pero normalmente será posible añadir alguna información más a h' , que puede obtenerse, por ejemplo, a partir de lo ya explorado. Además, como se comprobará a continuación, la elección de h' no puede influir nunca de modo negativo en la ε -admisibilidad del algoritmo $A\varepsilon^*$.

Teorema 9.14. *Si h es admisible, $A\varepsilon^*$ es ε -admissible.*

Demostración. Se probará que si el problema tiene solución, el algoritmo $A\varepsilon^*$ obtiene una solución ε -óptima. Puesto que el algoritmo siempre para (por el hecho de que el coste a través de cualquier camino infinito crece de forma no acotada), bastará demostrar que cuando es seleccionado un nodo terminal t , se verifica $f(t) \leq (1 + \varepsilon)C^*$.

Sea n_0 el nodo de *ABIERTA* con menor valor de f en el momento de la expansión del nodo objetivo t . Obsérvese que ese nodo no tiene por qué ser t pues se ha modificado la estrategia de A^* y ya no se expande necesariamente el nodo de menor coste estimado. Por otra parte, sea n_1 el nodo de un camino óptimo P desde el inicial a los objetivos que está en ese momento en *ABIERTA* y que sus antecesores a través de P ya fueron expandidos. En estas condiciones, se tiene que

$$\begin{aligned} f(t) &\leq f(n_0)(1 + \varepsilon) \leq f(n_1)(1 + \varepsilon) = (g^*(n_1) + h(n_1))(1 + \varepsilon) \\ &\leq (g^*(n_1) + h^*(n_1))(1 + \varepsilon) = (1 + \varepsilon)C^* \end{aligned}$$

□

Comparación entre los dos algoritmos. El algoritmo de ponderación dinámica es más simple ya que solamente tiene una lista de estados y un heurístico, pero tiene el inconveniente de que es necesario conocer la profundidad de la solución óptima, o al menos una buena cota superior. Por otra parte, $A\varepsilon^*$, al utilizar un segundo heurístico h' , que no tiene que ser admisible, ofrece un mecanismo más flexible del que ofrecen las funciones g y h , para hacer estimaciones del coste computacional de la búsqueda en una determinada dirección.

9.3.5 Diseño sistemático de heurísticos

Tal y como se ha comentado anteriormente, el diseño de buenos heurísticos es un proceso no trivial que requiere de una buena dosis de imaginación, y en el que se pueden utilizar todo tipo de informaciones e intuiciones sobre el dominio del problema. Siguiendo esta idea, en la sección 9.3.3 se han diseñado algunos heurísticos para cuatro problemas concretos, y además se ha justificado si se trataba o no de heurísticos admisibles. No obstante, para problemas más sofisticados, se puede intuir que el diseño de heurísticos puede ser inabordable si no se dispone de un método más sistemático que nos permita de alguna manera controlar la admisibilidad y la calidad de los heurísticos.

En el caso del problema del 8-puzzle, se ha diseñado, entre otros, el heurístico h_2 siguiendo este método intuitivo y luego se ha empleado un razonamiento para probar su admisibilidad que consiste en lo siguiente: dada una situación del tablero, para llegar al objetivo, cada ficha deberá seguir una trayectoria a través de sucesivas posiciones cada una de ellas adyacente ortogonalmente a la anterior. En el caso más favorable, esta trayectoria coincidirá con la distancia orthogonal desde la posición actual de la ficha hasta su posición en el objetivo. Esto es evidente porque no hay otra trayectoria con estas características que sea más corta.

Análogamente se podría haber llegado a este mismo heurístico mediante el siguiente razonamiento. Si se pudiese mover una ficha a cualquier posición adyacente orthogonalmente, aunque esta posición no estuviese vacía, entonces se podría llevar cada una de las fichas desde su posición actual hasta su posición en el objetivo con independencia de las otras fichas y, para cada una de ellas, se tendría que realizar solamente un número de movimientos igual a su distancia orthogonal a la posición que le corresponde en el objetivo.

Con el razonamiento anterior, lo que se ha hecho en realidad es considerar una versión simplificada del problema en la que se prescinde de la restricción de que la casilla de destino debe estar vacía. El uso de modelos simplificados es la base de muchas estimaciones que se hacen para resolver problemas cotidianos. Por ejemplo, para desplazarse en coche desde un punto a otro en una ciudad en la que no se conoce el sentido de circulación de las calles, inmediatamente se piensa en la distancia en línea recta entre los dos puntos y se inicia el desplazamiento en la dirección que más se aproxima a la que indica esta línea.

En esta sección se introduce un método sistemático de diseño de heurísticos basado en esta idea: el método de la relajación del problema. Los fundamentos de este método se describen en el texto de J. Pearl [Pearl, 1984, capítulo 4]. En líneas generales el método consiste en considerar una versión simplificada o relajada del problema original, de modo que esta versión simplificada se pueda resolver de forma óptima mediante un algoritmo polinomial. Luego, el coste óptimo del problema relajado se toma como una estimación del coste del problema original. Lógicamente, esta estimación es optimista ya que todas las soluciones del problema real son también soluciones del problema relajado, pero no al revés.

Para la aplicación sistemática del método, el primer paso consiste en enunciar el problema de modo que todas las restricciones queden reflejadas de forma explícita. En el siguiente paso lo que se debe hacer es considerar las posibles relajaciones que consisten en suponer que se relaja un subconjunto de las restricciones del problema hasta conseguir una versión del problema suficientemente simple para que se pueda resolver con un algoritmo polinomial. Además, debe tenerse en cuenta que cuanto menos se relaje el problema, mejor será el heurístico resultante ya que más se parecerá el problema relajado al problema original. En la sección siguiente, se ilustra este método con la aplicación a varios ejemplos ya conocidos.

Ya se ha comentado que los heurísticos obtenidos por el método de relajación del problema son admisibles. Ahora se probará que también son monótonos.

Teorema 9.15. *Todo heurístico h obtenido por el método de la relajación del proble-*

ma es monótono.

Demostración. Dados dos nodos cualesquiera, n y n' , basta con observar que una forma de resolver el problema n en el modelo relajado es ir hasta n' aplicando una regla de coste $c(n, n')$ y luego resolver n' en el modelo relajado. Por lo tanto, la mejor forma de resolver n en el modelo relajado es mejor o igual que ésta que consiste en ir a través de n' . Como $h(n)$ y $h(n')$ representan el coste de la mejor forma de resolver n y n' respectivamente en el modelo relajado, se tiene que

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

luego h es monótono. □

Este resultado tiene varias consecuencias inmediatas. En primer lugar pone de manifiesto la eficacia del método, ya que se trata de un método sistemático que proporciona heurísticos con buenas propiedades; es decir, no es preciso probar formalmente estas propiedades una vez obtenidos los heurísticos, ya que éstas son una consecuencia directa de la aplicación correcta del método. Por otra parte, pone de manifiesto que es el método que tienden a utilizar las personas de forma inconsciente, ya que la experiencia demuestra que la mayoría de los heurísticos admisibles que se diseñan razonando de una forma lógica son también monótonos. Piénsese, por ejemplo, en los heurísticos que se han visto para el problema del 8-puzzle, en particular en el heurístico h_3 .

9.4 Búsqueda con memoria limitada

El principal problema que tiene el algoritmo A* es el requerimiento de memoria que crece de forma exponencial con la profundidad, aunque dispongamos de buenos heurísticos. Para tratar de salvar este problema, se han propuesto variantes que incorporan mecanismos para limitar la cantidad de memoria. El primero de estos métodos se denomina IDA* (Iterative Deepening A*) y es una extensión del método de búsqueda iterativa en profundidad. El segundo se denomina SMA* (Simplified Memory-Bounded A*) y es similar a A*, pero restringe el tamaño de *ABIERTA* a un valor máximo prefijado. Para simplificar la exposición, se consideran versiones de estos algoritmos para el recorrido de árboles.

9.4.1 Algoritmo IDA*

Este algoritmo fue propuesto por R. Korf [Korf, 1985]. En principio, se establece una longitud² límite para la búsqueda igual al valor de $f(\text{initial})$ que es, obviamente, una cota inferior de la solución óptima, supuesto que el heurístico es admisible. En cada iteración, IDA* busca con una estrategia en profundidad descartando los nodos n cuya estimación $f(n)$ supere la longitud límite. Si en una iteración no se encuentra

²En este caso la longitud de un camino se refiere a la suma de los costes de sus arcos y no al número de éstos.

solución, se realiza una nueva iteración en la que la búsqueda comienza otra vez desde el principio, pero esta vez con una nueva longitud límite, el menor valor de f de los nodos descartados en la iteración anterior. Puesto que el algoritmo dispone de una función heurística h , se puede utilizar esta información para ordenar los sucesores del nodo expandido antes de insertarlos en *ABIERTA*, de modo que se consideren antes los sucesores más prometedores de cada nodo expandido. De este modo, si en una iteración se encuentra la solución, el número de nodos expandidos en esta iteración puede ser menor. Si por el contrario en una iteración no se encuentra solución, el número de nodos expandidos es el mismo que si los sucesores se ordenasen de otro modo. A continuación se muestran el algoritmo IDA* (9.3) y el algoritmo de búsqueda heurística con longitud limitada que es utilizado por el algoritmo IDA* en cada iteración (9.4).

Algoritmo 9.3 Algoritmo IDA* (Iterative Deepening A*).

```
LongitudLímite = f(inicial);
Solución = Primero en Profundidad con Longitud Limitada;
mientras (Solución = "no encontrado") hacer
    ProfundidadLímite = NuevaLongitudLímite;
    Solución = Primero en Profundidad con Longitud Limitada;
fin mientras
devolver Solución;
```

No es difícil comprobar que IDA* es admisible si h es monótono (si h es solamente admisible IDA* también es admisible, pero el razonamiento es más complejo), ya que si en una iteración no se encuentra solución, el nuevo límite de profundidad, calculado como el menor valor de f de los nodos descartados en esa iteración, es una cota inferior de la solución óptima, ya que los valores de f de los nodos a lo largo de una rama del árbol de búsqueda forman una secuencia no decreciente por ser h monótono. Luego si en la siguiente iteración se encuentra una solución, su coste debe ser igual al valor límite y por lo tanto la solución es óptima.

Puesto que el algoritmo realiza búsquedas en profundidad, el consumo de memoria es proporcional a la profundidad de la solución d y al factor de ramificación b , como se ha visto en la sección 8.4.1.4. Sin embargo, el tiempo de búsqueda es exponencial en la profundidad, es decir que en cada iteración tenemos una búsqueda con tiempo exponencial en la profundidad límite. En la práctica, si todos los nodos tienen valores de f distintos, en cada iteración solamente se añade un nodo al espacio de búsqueda y en este caso, si el algoritmo A* necesita expandir N nodos para llegar a una solución, el algoritmo IDA* necesita expandir $1 + 2 + \dots + N = O(N^2)$ nodos. Evidentemente si N es mucho espacio, N^2 también es mucho tiempo. Una forma de atajar este inconveniente es, en cada iteración, añadir un factor ε a la profundidad límite, con lo que se obtiene un algoritmo ε -admisible. Sin embargo, si el valor de f solamente cambia cada cierta profundidad, el tiempo requerido para una iteración es mucho mayor que el de la iteración anterior, de modo que el tiempo total de la búsqueda es prácticamente el tiempo de la última iteración realizada, por lo tanto similar al tiempo de A*.

Algoritmo 9.4 Algoritmo de búsqueda heurística en profundidad en árboles con longitud limitada.

```
ABIERTA = (inicial);
mientras (NoVacía(ABIERTA)) hacer
    n = ExtraePrimero(ABIERTA);
    si (EsObjetivo(n)) entonces
        devolver Camino(inicial, n)
    fin si
    S = ∅ ;
    si (f(n) < LongitudLímite) entonces
        S = Sucesores(n);
    si no
        Actualiza NuevaLongitudLímite con f(n);
    fin si
    si (Vacía(S)) entonces
        LimpiarTABLA_A(n)
    fin si
    para cada q de S hacer
        pone q en la TABLA_A con
            Anterior(q) = n,
            g(q) = Coste(inicial, n) + Coste(n, q),
            h(q) = Heurístico(q);
    fin para
    Ordena S según los valores de f de menor a mayor;
    ABIERTA = Concatenar(S, ABIERTA);
fin mientras
devolver "no encontrado" y NuevaLongitudLímite;
```

9.4.2 Algoritmo SMA*

El principal problema del algoritmo IDA* es la cantidad de nodos que tiene que volver a expandir, debido a que lo único que recuerda de una iteración a la siguiente es el valor del menor valor de f de los nodos descartados. El algoritmo SMA* propuesto por S. Russell [Russell, 1992] resuelve este problema a la vez es capaz de operar con una cantidad limitada de memoria. Para mantener una cierta uniformidad con las descripciones de los algoritmos anteriores, la descripción que se sigue aquí, y que se muestra en el Algoritmo 9.5, difiere en algunos aspectos de la original que puede verse también en el texto de Russell y Norvig [Russell y Norvig, 2003].

En el Algoritmo 9.5, el valor de la variable *Límite* se refiere al número máximo de nodos que se pueden almacenar en la *TABLA_A*, y por lo tanto a la profundidad máxima de un camino registrado en esta tabla. La llamada a la función *SiguienteSucesor(n)* proporciona en cada llamada un sucesor del nodo, en principio en un orden no determinado, por lo que es preciso algún mecanismo de control de los sucesores generados hasta el momento. Este mecanismo se puede apoyar en la *TABLA_A*. En líneas generales, el funcionamiento del algoritmo SMA* es el siguiente. Cuando necesita expandir un nodo y no tiene espacio en la *TABLA_A*, elimina un nodo de esta tabla y de *ABIERTA*. Estos son los llamados nodos olvidados y son aquellos menos prometedores, es decir los que tienen un mayor valor de f en *ABIERTA*. El algoritmo recuerda en cada nodo el mejor f de los hijos de ese nodo

que han sido olvidados. Para no volver a explorar subárboles que han sido eliminados de memoria, SMA* mantiene en los nodos ancestros información acerca de la calidad del mejor camino en cada subárbol descartado. De este modo, solamente reexplora un subárbol descartado cuando el resto de posibilidades es peor de acuerdo con las estimaciones. Para proceder de este modo, cuando se descartan todos los sucesores de un nodo n , el algoritmo recuerda la estimación del mejor camino a través de n .

El algoritmo SMA* es bastante sofisticado, por lo que una forma de entenderlo es a través de un buen ejemplo como el que se puede encontrar en el texto de Russell y Norvig [Russell y Norvig, 2003, pp. 108-109].

Algoritmo 9.5 Algoritmo SMA* (Simplified Memory-Bounded A*).

```

Inserta el  inicial en ABIERTA y en la TABLA_A;
mientras (NoVacia(ABIERTA)) hacer
     $n = \text{Primer}(ABIERTA);$ 
    si (EsObjetivo(n)) entonces
        devolver Camino(inicial, n)
    fin si
     $s = \text{SiguienteSucesor}(n);$ 
    si (NoEsObjetivo(s) y Profundidad(s) = Límite) entonces
         $f(s) = \infty;$ 
    si no
         $f(s) = \max(f(n), g(s) + h(s));$ 
    fin si
    si Todos los sucesores de  $n$  han sido generados entonces
        Actualiza  $f(n)$  y el  $f$  de los ancestros de  $n$ 
    fin si
    si (Todos los sucesores de  $n$  están en la TABLA_A) entonces
        Elimina  $n$  de ABIERTA;
    fin si
    si (la TABLA_A está llena) entonces
        Elimina de ABIERTA y de la TABLA_A el nodo con mayor  $f$  en ABIERTA y menor
        profundidad, actualizando la estimación del mejor nodo hijo olvidado en el padre del nodo
        eliminado;
        Inserta el padre del nodo eliminado en ABIERTA si no está;
    fin si
    Inserta  $s$  en ABIERTA y en la TABLA_A;
fin mientras
devolver “no solución”;
```

Por último, tal y como se indica en el texto de Russell y Norvig, las propiedades principales del algoritmo SMA* son las siguientes:

- Es capaz de evolucionar utilizando la memoria que tenga disponible.
- Evita estados repetidos en la medida en que la cantidad de memoria disponible se lo permita.
- Es completo si la memoria disponible es suficiente para almacenar el camino a la solución menos profunda.
- Es admisible si tiene suficiente memoria para almacenar el camino hasta la

solución óptima menos profunda. En otro caso, devuelve la mejor solución que se puede alcanzar con la memoria disponible.

- Si la memoria es suficiente para el árbol de búsqueda completo, la eficiencia de la búsqueda es óptima.

Sin embargo, no está resuelta la cuestión de si SMA* tiene una eficiencia mejor o igual que cualquier otro algoritmo, dada la misma información heurística y la misma cantidad de memoria.

9.5 Algoritmos voraces

Los algoritmos voraces (greedy) constituyen una alternativa a la búsqueda exhaustiva. La idea básica es que un algoritmo de esta clase toma decisiones de forma irrevocable, de modo que nunca considera otra alternativa. Una consecuencia de este modo de proceder es que los algoritmos voraces no son admisibles, y en general tampoco son completos. Pero por otra parte resultan extremadamente eficientes, por lo que son muy adecuados por ejemplo para los sistemas de tiempo real. Además si la decisión irrevocable está guiada por un buen heurístico, la probabilidad de obtener una buena solución aumenta. Estos algoritmos aparecen con profusión en la literatura, por ejemplo los algoritmos de planificación del procesador, u otro tipo de recursos, de los sistemas operativos, o los algoritmos de enrutamiento de mensajes en redes de comunicaciones, suelen ser algoritmos de este tipo.

El diseño de un algoritmo voraz para la búsqueda en espacios de estados es simple a partir de los algoritmos que se han visto hasta aquí; no obstante dada su simplicidad lo lógico es realizar una implementación más sencilla y eficiente. Por ejemplo, a partir de un algoritmo A*, si en cada expansión se prescinde de todos los sucesores salvo del más prometedor de acuerdo con el heurístico h , se tiene un algoritmo voraz. En este caso se trata de un algoritmo determinista. Si, además, los empates se resuelven de forma aleatoria, se tiene un algoritmo no determinista que, si se ejecuta varias veces, proporciona distintas soluciones con una calidad media que dependerá de la precisión de las estimaciones del heurístico. La variedad de las soluciones se puede aumentar si en lugar de elegir aleatoriamente entre los mejores sucesores con una distribución uniforme, la elección se hace con una distribución proporcional a la bondad estimada de cada sucesor.

Los algoritmos voraces son útiles en la práctica para resolver muchos problemas, y se pueden utilizar además en combinación con otras estrategias de búsqueda, como por ejemplo los esquemas de ramificación y poda que se describen en la sección siguiente, o los algoritmos evolutivos que se verán en el capítulo 11.

9.6 Algoritmos de ramificación y poda

Los algoritmos de ramificación y poda constituyen, posiblemente, el método más empleado en la resolución de problemas de optimización combinatoria. Como se verá,

tienen muchos elementos en común con otros algoritmos vistos anteriormente, como el propio A*, pero tienen también algunas características propias. En primer lugar, con los algoritmos anteriores, la búsqueda se plantea como un proceso en el que la solución se va construyendo paso a paso, de modo que un camino desde el inicial a un estado representa parte de la solución del problema y el estado representa un subproblema que se debe resolver para llegar a una solución del problema original. Por ejemplo, el estado inicial se interpreta como una situación en la que no se ha hecho nada todavía y por lo tanto en la que el subproblema a resolver es el problema original. Sin embargo, en un algoritmo de ramificación y poda, el planteamiento es otro. Cada estado se interpreta como un subconjunto de soluciones del problema original, en el estado inicial se tienen todas las soluciones y a partir de aquí, mediante el proceso de ramificación, un conjunto de soluciones se descompone en la unión disjunta de varios conjuntos, hasta llegar a conjuntos unitarios que representan soluciones del problema. Este planteamiento lleva asociado que el espacio de búsqueda tenga estructura de árbol. Históricamente, los algoritmos de ramificación y poda han sido más utilizados por la comunidad de Investigación Operativa, mientras que los algoritmos de búsqueda tipo A* lo han sido por la comunidad de IA.

El Algoritmo 9.6 muestra el esquema de un algoritmo de ramificación y poda. Se trata de una versión que está inspirada en propuestas concretas, como el algoritmo de P. Brucker para resolver problemas de scheduling, en [Brucker, 2004] o el algoritmo de C. Artigues y D. Feillet también para problemas de scheduling [Artigues y Feillet, 2007], aunque evidentemente es una versión general y por lo tanto aplicable a todo tipo de problemas de optimización combinatoria.

Los algoritmos de ramificación y poda tienen cuatro componentes fundamentales: un esquema de ramificación, un método de cálculo de cotas inferiores (Lower Bounds), un método de cálculo de cotas superiores (Upper Bounds) y una estrategia de control.

Esquema de ramificación. La ramificación de un nodo es equivalente al cálculo de sucesores en los algoritmos de búsqueda anteriores. Cada nodo representa un problema y dado un nodo n , $Y(n)$ representa al conjunto de todas las soluciones de n . Obviamente este conjunto nunca se representa por extensión. La ramificación de n consiste en descomponer el problema n en una serie de subproblemas q_1, \dots, q_p , de forma que $Y(n)$ es la unión disjunta de $Y(q_1), \dots, Y(q_p)$, es decir $Y(n) = Y(q_1) \cup \dots \cup Y(q_p)$, y $Y(q_i) \cap Y(q_j) = \emptyset$, $1 \leq i, j \leq p$, $i \neq j$. El estado *inicial* es el nodo raíz del árbol de búsqueda, y a partir de él se obtiene el resto del árbol de búsqueda mediante la aplicación recurrente del esquema de ramificación. Los nodos hoja de este árbol se caracterizan por el hecho de que representan una única solución del problema.

Cálculo de cotas inferiores. El cálculo de cotas inferiores para un nodo n es análogo al cálculo de heurísticos admisibles. La obtención de buenas cotas inferiores es fundamental para la eficiencia del algoritmo de ramificación y poda. De la misma forma que los heurísticos bien informados permiten expandir menos nodos al algoritmo A*, mediante las cotas inferiores se pueden realizar procesos de poda que permiten descartar subárboles del árbol de búsqueda. Dado un nodo n el algoritmo de cálculo de cotas inferiores produce una cota inferior de la mejor solución de $Y(n)$ (el valor

Algoritmo 9.6 Esquema de un algoritmo de ramificación y poda.

```
ABIERTA = inicial;
UB = ∞;
s = ∅;
LB = cota inferior de inicial;
mientras (No Vacía(ABIERTA) y (LB < UB) y (No CriterioTerminación)) ) hacer
    Extrae un elemento n de ABIERTA;
    sn = solución de Y(n) obtenida mediante un algoritmo heurístico;
    si (Coste(sn) < UB) entonces
        s = sn;
        UB = valor de sn;
    fin si
    Calcula los sucesores de n, q1, ..., qp, ordenados con algún criterio, de modo que Y(n) = Y(q1) ∪
    ... ∪ Y(qp), siendo esta unión disjunta;
    Calcula cotas inferiores LBi de qi, 1 ≤ i ≤ p;
    para cada i de 1 a p hacer
        si (LBi < UB) entonces
            si (|Y(qi)| = 1) entonces
                UB = LBi;
                s = la solución de Y(qi);
            si no
                Añade qi a ABIERTA;
            fin si
        fin si
    fin para
    LB = min (LBq, q ∈ ABIERTA);
fin mientras
devolver s, UB, LB;
```

de esta cota inferior es análogo al valor $f(n) = g(n) + h(n)$ en A* si tenemos un heuristicó admisible). Obviamente, esta cota inferior no tiene por qué ser cota inferior del problema, ya que la mejor solución del problema puede no estar incluida en $Y(n)$. Sólo cuando el algoritmo se aplica al nodo *inicial* se puede tener la seguridad de que se obtiene una cota inferior de la solución óptima del problema. En la descripción del Algoritmo 9.6, la condición $|Y(q_i)| = 1$ indica que el nodo q_i representa una única solución y por lo tanto la función del cálculo de la cota inferior es capaz de calcular el valor de la única solución del conjunto $Y(q_i)$.

Cálculo de cotas superiores. Las cotas superiores están asociadas a soluciones reales del problema completo. El algoritmo de ramificación y poda mantiene en una variable *UB* la menor de las cotas superiores obtenidas hasta el momento, y la solución correspondiente en la variable *s*. Típicamente, los algoritmos de ramificación y poda obtienen estas cotas cada vez que en la búsqueda se llega a un nodo hoja. Sin embargo, la obtención de cotas superiores se puede ampliar con el cálculo de cotas para cada uno de los nodos que se van expandiendo mediante un algoritmo heurístico. Se trata simplemente de utilizar un algoritmo que para un nodo *n* produzca una buena solución de $Y(n)$. Este algoritmo debe ser muy eficiente, por lo que suele ser un algoritmo voraz, como los descritos en la sección 9.5.

Estrategia de control. La estrategia de control se refiere a la gestión de la lista *ABIERTA*, que contiene los nodos candidatos a ser desarrollados mediante el esquema de ramificación, es decir a la estrategia que se utiliza para decidir qué nodos entran en *ABIERTA* y cuál es el siguiente nodo expandido. Generalmente, por razones de eficiencia los algoritmos de ramificación y poda utilizan una estrategia de búsqueda en profundidad, es decir *ABIERTA* se trata como una pila. Como además se dispone de la información que proporcionan las cotas inferiores, se pueden ordenar los sucesores de cada nodo antes de insertarlos en *ABIERTA*. Sin embargo, se pueden utilizar otras posibilidades. Por ejemplo, *ABIERTA* puede estar ordenada de acuerdo con las cotas inferiores y en ese caso tendríamos una estrategia análoga a la de A*.

Por último, al disponer de una cota superior en la variable *UB*, si después de la ramificación de un nodo, uno de sus sucesores q_i tiene una cota inferior $LB_i \geq UB$, ya se conoce que en el conjunto $Y(q_i)$ no hay una solución mejor que la que ya se tiene en este momento, y en consecuencia q_i no se inserta en *ABIERTA*. Este es el proceso de poda que permite reducir el espacio de búsqueda efectivo y que será tanto más eficiente cuanto mejores sean las estimaciones que se van haciendo mediante las cotas inferiores y superiores.

Para asegurar la condición de admisibilidad, los algoritmos de ramificación y poda deben recorrer todo el espacio de búsqueda, salvo las partes que se podan, antes de terminar, ya que, en general, la primera solución que encuentran no es la óptima. De ese modo, el criterio de parada será en principio que *ABIERTA* se agote. No obstante, se puede mantener una cota inferior del problema, en la variable *LB*, que se calcula como el mínimo de las cotas inferiores, LB_q , de los nodos q contenidos en *ABIERTA* justo antes de cada expansión. De ese modo, si se encuentra una solución cuyo valor coincide con la cota inferior, es decir $LB = UB$, se tiene la seguridad de que se trata de una solución óptima y se puede detener la búsqueda. Si la instancia del problema es difícil de resolver y en consecuencia da lugar a un espacio de búsqueda muy grande, el tiempo puede resultar prohibitivo. En este caso es posible detener la búsqueda mediante algún criterio adicional, por ejemplo después de un cierto número de nodos expandidos, o después de un determinado tiempo de ejecución. Para ello se puede utilizar un tercer criterio de parada. Obviamente, en este caso se pierde la admisibilidad, pero el algoritmo devolverá la mejor solución encontrada, con su coste *UB*, y la mejor cota inferior, *LB*, obtenida durante la búsqueda. Esta es una característica importante de este tipo de algoritmos: son admisibles si disponen de un tiempo de ejecución suficiente, pero producen una solución y una cota inferior si el tiempo es limitado.

9.7 Algoritmos de mejora iterativa o búsqueda local

En general, la solución que proporciona un algoritmo de búsqueda viene dada por un nodo objetivo junto con un camino desde el nodo inicial hasta este nodo. Sin embargo, hay muchos problemas para los cuales el nodo solución contiene toda la información necesaria y en este caso el camino es irrelevante. Este es el caso del TSP y del problema de las *N*-reinas. Mientras que para otros problemas, como el 8-puzzle

y el problema planificación de actuaciones de robots, la información importante es el camino desde el estado inicial a la solución. Para problemas como el TSP, es posible plantear otro tipo de búsqueda que consiste en partir de un estado que representa una solución del problema y luego hacer pequeños cambios en la configuración de este estado buscando mejoras de la solución. Esta es la idea de los métodos de mejora iterativa, que se suelen denominar también métodos de búsqueda local por el hecho de que con los cambios que se realizan en la configuración de un estado, en realidad lo que se hace es generar configuraciones próximas o vecinas a esta configuración.

La estructura básica de un método de búsqueda local es la que se muestra en el Algoritmo 9.7. El algoritmo parte de una solución inicial y en cada iteración calcula un conjunto de soluciones vecinas mediante una regla de vecindad. Cada una de estas soluciones debe ser evaluada, siendo ésta una de las acciones más críticas del algoritmo por el elevado tiempo de ejecución que puede suponer, especialmente si el número de soluciones vecinas es muy grande, o bien si el procedimiento de evaluación de cada solución es muy costoso. A continuación, se selecciona una de las soluciones vecinas con un determinado criterio, normalmente la que tiene menor coste. Si esta solución cumple el criterio de aceptación, que puede ser simplemente que sea mejor que la solución actual S , la solución seleccionada reemplaza a la solución S , y el proceso continúa hasta que se cumpla el criterio de finalización. Este criterio suele ser que se agote un determinado número de iteraciones, o bien que no se produzcan mejoras en los últimos intentos. La regla de vecindad debe cumplir la propiedad de conectividad, es decir que desde cualquier estado se pueda alcanzar un objetivo óptimo mediante una secuencia de transformaciones. Además, las reglas suelen ser simétricas, es decir que si una solución x es vecina de otra y , y también es vecina de x .

El diseño de un algoritmo de búsqueda local para un problema como el TSP es bastante simple. Si el grafo de conexiones está totalmente conectado, entonces cualquier permutación de las ciudades, prescindiendo de la ciudad de partida, es una solución del problema. El coste de esta solución se calcula de forma inmediata en un tiempo del orden de $O(N)$, siendo N el número de ciudades. El cálculo de soluciones vecinas se puede hacer de varias formas, la más simple consiste en mover cada ciudad a continuación de la siguiente en la permutación, y la última por delante de la primera; así se obtienen N soluciones vecinas. Estas soluciones se pueden evaluar sin necesidad de recorrer toda la secuencia de ciudades, ya que en cada solución vecina solamente cambian tres arcos con respecto a la solución original.

Algoritmo 9.7 Esquema de un algoritmo de búsqueda local.

```
 $S = SolucionInicial;$ 
mientras (no CriterioDeTerminación) hacer
   $V = SolucionesVecinas(S);$ 
  EvaluarSoluciones(V);
   $S_1 = Seleccion(V);$ 
  si (CriterioDeAceptación) entonces
     $S = S_1;$ 
  fin si
fin mientras
devolver  $S;$ 
```

En el caso del problema de las N -reinas, se puede codificar una solución mediante una distribución de las N -reinas de forma que no haya dos reinas en una misma fila. Pero una configuración así puede no ser solución del problema ya que pueden estar dos o más reinas en una misma columna o en una misma diagonal; una configuración de estas características se considera una solución potencial del problema, y esta solución potencial se parecerá tanto más a una solución real cuantas menos reinas se ataquen entre sí. Así, se puede dar a cada solución potencial un coste proporcional al número de reinas que son atacadas. Un regla de vecindad simple consiste en mover cada reina en su fila a una posición distinta, con lo que tendremos $(N - 1)N$ vecinos. La evaluación de cada vecino se puede hacer en un tiempo del orden de $O(N^2)$ ya que tendremos que comprobar si cada reina es atacada por alguna de las demás.

Los algoritmos de búsqueda local realizan una búsqueda en un espacio de soluciones potenciales del problema tratando de encontrar aquella que maximice, o minimice, una determinada función objetivo. Esta función objetivo tiene normalmente una forma o relieve (que se suele denominar *landscape*) muy irregular, con numerosos máximos y mínimos locales, por lo que la búsqueda del óptimo global es muy costosa. Por este motivo la búsqueda se suele asociar metafóricamente con una excursión a través de una cordillera en la que se trata de alcanzar la cima más alta. Normalmente, la búsqueda local no mantiene traza de todo el proceso realizado hasta el momento, sino que solamente utiliza información sobre el estado actual y sus vecinos. Siguiendo la metáfora anterior, tal y como apuntan Russell y Norvig, es como caminar por una cordillera sin conocerla, en un día de niebla espesa y, además, sufriendo amnesia. A pesar de esto, la búsqueda local es una estrategia muy eficaz en muchos casos que también se puede combinar con otras estrategias, como por ejemplo los algoritmos evolutivos que se describen en el capítulo 11. En las secciones siguientes se muestran tres variantes de búsqueda local de uso muy extendido: los algoritmos de escalada o máximo gradiente, la búsqueda tabú y el temple simulado (o *simulated annealing*).

9.7.1 Algoritmos de escalada o máximo gradiente

En la búsqueda por máximo gradiente el criterio de aceptación de la solución vecina S_1 , frente a la solución actual S , es que S_1 sea mejor o igual que S . Previamente en la selección se intenta elegir una solución que mejore a S , o bien la mejor de todas las soluciones de la vecindad. Este tipo de búsqueda es simple, pero tiene algunos inconvenientes, concretamente tiene problemas en las siguientes situaciones:

- *Óptimos locales*: cuando se alcanza uno de estos puntos del espacio de búsqueda, todos los vecinos son peores con lo que la búsqueda finaliza sin encontrar el óptimo global.
- *Regiones planas*: en este caso todos los vecinos tienen el mismo valor que la solución actual y la búsqueda es aleatoria.
- *Crestas*: si hay una cresta con pendientes muy marcadas, puede ser fácil alcanzar la cima de la cresta. Sin embargo, si la pendiente de la cima es muy suave en la dirección del óptimo global, resulta difícil guiar la búsqueda a través de la cima sin desviarse hacia los lados de la cresta; a menos que existan operadores específicos para generar vecinos a través de la cima.

En estos tres casos lo que ocurre es que la búsqueda se queda estancada en un óptimo local, que puede ser una solución razonable o no serlo. Lo que se puede hacer en estos casos es reiniciar la búsqueda a partir de otra situación de partida, lo que se suele denominar *búsqueda multiarranque*, con lo que se alcanzaría posiblemente otro óptimo local distinto. Así, después de un número de intentos se podría llegar a una solución aceptable, sobre todo si el número de óptimos locales es pequeño. Sin embargo, para los problemas NP-duros, el número de óptimos locales suele ser un número exponencial, con lo que las posibilidades de alcanzar un óptimo global son muy escasas.

9.7.2 Temple simulado

La idea de este método consiste en admitir, con una cierta probabilidad, algunas transiciones en las que la solución actual empeore; de este modo se puede salir de óptimos locales. El Algoritmo 9.8 muestra el método de temple simulado. Las diferencias principales de este algoritmo con respecto al algoritmo de escalada son las siguientes. En el temple simulado se realiza una selección aleatoria entre los vecinos del estado actual. Si esta solución es mejor que la actual, se realiza la transición de forma incondicional, como en el caso del algoritmo de escalada. Pero si la solución vecina es peor que la actual, entonces la nueva solución se acepta con una determinada probabilidad que depende de dos parámetros: la temperatura T y el incremento de energía ΔE . Los nombres de estos parámetros se han elegido por la analogía del método con el proceso físico de enfriamiento de un líquido hasta que se solidifica.

Al principio de la búsqueda, la temperatura tiene un valor alto, inicialmente T_0 , de modo que la probabilidad de aceptar una solución peor que la actual es alta. A medida que la búsqueda progresá, el valor de T se va actualizando de forma $T = \alpha(t, T)$, siendo t la iteración actual y α una función que decrece al aumentar t . De este modo, la probabilidad de aceptar una solución que empeora a la actual va disminuyendo a medida que avanza la búsqueda, hasta que al final prácticamente sólo se admiten soluciones que mejoren o igualen a la actual.

Algoritmo 9.8 Esquema de un algoritmo de temple simulado.

```
 $S = SolucionInicial;$ 
 $t = 0;$ 
 $T = T_0;$ 
mientras (no CriterioDeTerminación) hacer
     $S_1 = SeleccionAleatoria(SolucionesVecinas(S));$ 
     $\Delta E = Coste(S_1) - Coste(S);$ 
    si ( $\Delta E < 0$ ) entonces
         $S = S_1;$ 
    si no
         $S = S_1$  con probabilidad  $e^{-\Delta E/T};$ 
    fin si
     $t = t + 1;$ 
     $T = \alpha(t, T);$ 
fin mientras
devolver  $S;$ 
```

La función α se puede definir de varias formas, dos posibilidades simples son las siguientes: $\alpha(t, T) = kT$, con $0 < k < 1$, y $\alpha(t, T) = T/(1 + kT)$, siendo $k > 0$ un valor muy pequeño. En general, si la variación de T es lenta y el valor de T_0 alto, la probabilidad de llegar a una buena solución es mayor, pero la búsqueda es más costosa ya que el criterio de terminación suele ser que T alcance un valor suficientemente pequeño.

El principal inconveniente que presenta este método es que requiere un ajuste de parámetros adecuado. Normalmente, este ajuste depende fuertemente del problema y hay que realizarlo de forma experimental. No obstante, el temple simulado es un método que resulta eficiente en muchos problemas.

9.7.3 Búsqueda tabú

La diferencia esencial de la búsqueda tabú con respecto a los métodos anteriores es que dispone de un mecanismo de memoria. Este mecanismo se utiliza para evitar la generación de algunos vecinos dependiendo de la historia reciente, o de la frecuencia con la que se realizaron algunas transformaciones para llegar al estado actual. Por ejemplo, si la regla de vecindad es simétrica, y se genera x como vecino de y , suele ser buena idea recordarlo y no generar y inmediatamente a partir de x .

En el caso más simple, el mecanismo de memoria se realiza mediante una estructura H que registra las transformaciones que dieron lugar a las últimas soluciones, de modo que estas transformaciones no se consideran en la generación de los vecinos de la solución actual, es decir se consideran tabú. Por ejemplo, en una instancia del problema TSP con 6 ciudades A, B, C, D, E y F, siendo A la ciudad de partida, si la solución actual viene dada por la permutación de las ciudades (C D B F E) y las tres últimas transformaciones realizadas fueron los intercambios (B,C), (B,D) y (E,F), la estructura H puede ser una lista tabú de tamaño 3 cuyo contenido actual es ((E,F) (B,D) (B,C)). De este modo, al generar los vecinos del estado (C D B F E), no se consideran los estados correspondientes a las transformaciones que indica la lista tabú, es decir (C D B E F) y (C B D F E). El tamaño de la lista tabú debe ajustarse en función de las características del problema.

En algunos casos, es posible que un movimiento de la lista tabú produzca en la solución actual una mejora. Por ejemplo, en el caso anterior el intercambio (B,D) podría estar en esta situación. Por este motivo se introduce lo que se denomina *criterio de aspiración* y que consiste en establecer excepciones a lo que indica la lista tabú, concretamente lo que se suele hacer es que si un movimiento tabú produce una solución que mejora a la mejor solución encontrada hasta el momento, este movimiento se aplica como si no estuviese en la lista.

Al igual que el método anterior, el principal inconveniente de la búsqueda tabú es el ajuste de parámetros, como el tamaño de la lista tabú, y la elección de los movimientos que se deben registrar y la definición del criterio de aspiración. También como en el caso del temple simulado, la búsqueda tabú se ha utilizado con éxito en muchos problemas, en particular cuando se combina con otras estrategias como por ejemplo los algoritmos evolutivos.

9.8 Resumen

En este capítulo se han visto técnicas que permiten introducir conocimiento específico sobre el dominio del problema en los algoritmos de búsqueda. Se ha hecho hincapié en los métodos más generales, como la búsqueda en espacios de estados y los algoritmos de mejora iterativa, y se ha visto cómo aplicar estos métodos a problemas formales. No obstante, todos estos métodos tienen aplicación a numerosos problemas reales que se presentan en muchos sectores de la sociedad. Cada vez se plantean problemas más complejos y de mayor tamaño que requieren de la aplicación de nuevos métodos de búsqueda en los que el conocimiento juega un papel fundamental para llegar a soluciones satisfactorias.

Lógicamente, existen otros métodos de resolución de problemas que no se han descrito en el capítulo, algunos de los cuales se describen en capítulos posteriores, como por ejemplo los métodos específicos de resolución de problemas de satisfacción restricciones que se tratan en el capítulo 10, o la computación evolutiva que se trata en el capítulo 11. Tampoco hemos tratado aquí los métodos de búsqueda en juegos con adversario, como el método minimax y el procedimiento de poda alfa-beta asociado, por tratarse de métodos de aplicación más restringida. Este tipo de sistemas de búsqueda se tratan con profundidad en otros textos como el de J. Pearl [Pearl, 1984, Parte III] o el de S. Russell y P. Norvig [Russell y Norvig, 2003, capítulo 5]. Incluso, dentro del paradigma de búsqueda en espacios de estados, hay en la literatura otras propuestas que han demostrado ser eficientes en muchos problemas. Por ejemplo, los métodos de búsqueda con discrepancia limitada (LDS) que se basan en considerar que el heurístico hace una buena elección un porcentaje alto de veces. En consecuencia, realizan una búsqueda visitando en primer lugar la rama del árbol correspondiente a la primera elección del heurístico, es decir la rama sin discrepancias heurísticas; a continuación consideran las ramas con una sola discrepancia, y así sucesivamente. Algunas de las propuestas más interesantes de este tipo de búsqueda se pueden ver en el artículo de P. Meseguer y T. Walsh [Meseguer y Walsh, 1998].

Recientemente se ha acuñado el término metaheurística para hacer referencia a técnicas generales de diseño de algoritmos heurísticos. Los métodos de búsqueda en espacios de estados, la búsqueda local y la computación evolutiva son un caso particular de metaheurísticas. No obstante, existen otras que no se han tratado aquí, pero que también son muy interesantes, como los algoritmos GRASP o la búsqueda dispersa. En el número 19 de la Revista Iberoamericana de Inteligencia Artificial, del año 2003, editada por la Asociación Española para la Inteligencia Artificial (AEPIA), se incluye una monografía sobre metaheurísticas que ofrece una visión global sobre estas técnicas. También, el texto de Z. Michalewicz y D. B. Fogel [Michalewicz y Fogel, 2000] trata con profundidad varias de estas técnicas, así como otros métodos de resolución de problemas basados en técnicas de programación lineal. Además, este texto trata de dar una guía sobre cómo distinguir si una determinada técnica es o no aplicable a un tipo de problemas; aunque, evidentemente, ésta es una capacidad que solamente se adquiere con la experiencia. En la sección siguiente se proponen algunos ejercicios sobre aspectos formales de los algoritmos y la aplicación de los mismos a otros problemas. El lector interesado puede encontrar también un

gran número de problemas resueltos en el texto de S. Fernández, J. González y J. Mira [Fernández y otros, 1998].

9.9 Ejercicios resueltos

En esta sección abordaremos el diseño sistemático de heurísticos, mediante la aplicación del método de la relajación del problema a varios ejemplos ya conocidos.

9.1. El problema del 8-puzzle

En este caso el problema puede enunciarse del siguiente modo.

Se trata de encontrar una secuencia de movimientos de longitud mínima para llegar desde una configuración inicial de 8 fichas en un tablero 3×3 hasta una configuración objetivo, de modo que cada movimiento elemental de una ficha desde una casilla A a otra casilla B cumpla las 3 restricciones siguientes:

- R₁. Las casillas A y B deben ser adyacentes.*
- R₂. Las casillas A y B deben estar en la misma fila o en la misma columna, es decir deben ser ortogonales.*
- R₃. La casilla B debe estar vacía.*

A partir de este enunciado se puede pensar en distintas relajaciones, en realidad tantas como el número de subconjuntos del conjunto de todas las restricciones. Se consideran en principio las dos siguientes.

Relajación 1: Se prescinde de las tres restricciones R₁, R₂, y R₃.

En este caso el problema relajado se resuelve de forma óptima sin más que mover cada ficha a la posición que le corresponde en el objetivo, ya que la posición de destino no tiene que ser adyacente, ni orthogonal a la posición origen, y tampoco tiene que estar vacía. De este modo se obtiene de nuevo el heurístico h₁, pero aplicando un método sistemático.

Relajación 2: Se relaja solamente de la restricción R₃.

En este caso el problema relajado se resuelve llevando cada ficha a su posición en el objetivo pero a través de una trayectoria de longitud mínima de movimientos a posiciones adyacentes orthogonalmente. Así se ha redescubierto h₂.

Dado que h₁ y h₂ se obtienen a partir de simplificaciones del problema, se puede asegurar que son monótonos y, por lo tanto, admisibles, sin necesidad de hacer ningún otro tipo de razonamiento, como el que se ha tenido que hacer en la sección 9.3.3.1 cuando se definieron estos mismos heurísticos mediante otro método. Además, como el conjunto de restricciones que se relajan en la *Relajación 2* es un subconjunto de las restricciones que se relajan en la *Relajación 1*, el problema resultante de la *Relajación*

1 en realidad es una versión relajada del problema resultante de la *Relajación 2*; en consecuencia se tiene que $h_2(n) \geq h_1(n)$ para todo nodo n , como ya se conocía también, pero ahora sin necesidad de realizar ninguna prueba adicional.

A modo de ejercicio se propone intentar otras relajaciones, por ejemplo prescindir solamente de R_1 , o de R_2 , o bien de cualquier subconjunto de dos de las tres restricciones. También puede resultar un ejercicio interesante tratar de llegar al heurístico h_3 definido en la sección 9.3.3.1, que es admisible pero no monótono, a través del método de la relajación del problema.

■

9.2. El problema del viajante de comercio (TSP). El problema del TSP también ofrece bastantes posibilidades de relajación para obtener distintos heurísticos. Para enunciar bien el problema, o mejor dicho el subproblema, se debe tener en cuenta que el subproblema que representa un estado, en el que ya se han visitado un subconjunto de las ciudades comenzando por la ciudad de partida, consiste en buscar un camino de coste mínimo que una a la última ciudad visitada con la ciudad de partida pasando una y solo una vez a través de cada una de las ciudades no visitadas. Si el problema tiene N ciudades, y en el estado n han sido visitadas k ciudades, sea G_n el grafo resultante de eliminar del grafo de conexiones todas las ciudades intermedias del estado, y la conexión de la ciudad A con la última. En la Figura 9.4 se muestra el grafo G_n para el estado (A C B) correspondiente al problema de la Figura 8.8 del capítulo 8.

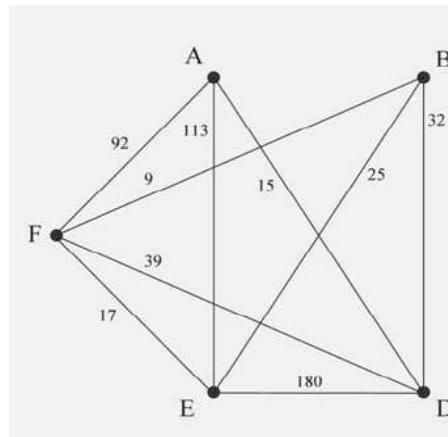


Figura 9.4: Subgrafo de conexiones G_n para el estado $n = (A C B)$ de la instancia del problema TSP de la figura 8.8 del capítulo 8.

El problema asociado a un nodo n se puede enunciar como sigue:

Se trata de calcular un subconjunto de ejes, del grafo G_n , de coste mínimo que satisfaga las restricciones siguientes:

- R₁. Que tenga $N - k + 1$ ejes.*
- R₂. Que todos los ejes sean distintos.*
- R₃. Que toque a todas las ciudades no visitadas de G_n .*
- R₄. Que toque a la ciudad de partida.*
- R₅. Que toque a la última ciudad visitada en el estado.*
- R₆. Que sea conexo.*
- R₇. Que sea de grado 1 para la primera y la última ciudad del estado, y de grado 2 para las ciudades no visitadas.*

Es evidente que cualquier subconjunto de arcos de G_n que satisfaga las siete restricciones anteriores es un camino que conecta la última ciudad visitada con la ciudad de partida. Si además es de coste mínimo, al ser añadido al camino representado por el estado n , proporciona un camino hamiltoniano de coste mínimo que se puede construir a partir de n , y en consecuencia es una solución del problema.

Si se observan las siete restricciones anteriores, se puede ver que no son independientes, por ejemplo, la restricción R_3 es una consecuencia de la restricción R_7 . Luego se podría haber prescindido de R_3 en la formulación del problema. Sin embargo, si se considera esta restricción, las posibilidades de relajación son mayores, ya que se puede relajar R_7 y mantener R_3 . Asimismo, las restricciones R_3 , R_4 y R_5 se podrían haber enunciado como una única restricción, pero de este modo habría menos posibilidades de relajación. Así, se pueden considerar en principio las relajaciones siguientes:

Relajación 1: Se prescinde de las cuatro restricciones R_2 , R_5 , R_6 y R_7 .

Con esta relajación, la solución óptima del problema se obtiene eligiendo para cada una de las ciudades no visitadas y para la ciudad de partida el eje de coste mínimo que toca a esa ciudad. Con lo que el problema relajado se resuelve en tiempo polinomial y da lugar al heuristicó h_1 definido como

$$h_1(n) = \text{suma de los costes de los ejes de coste mínimo que salen de las ciudades no visitadas y de la ciudad de partida.}$$

Para el nodo $n = (\text{A C B})$ se tiene que $h_1(n) = 15 + 17 + 9 + 15 = 56$.

¿Qué pasaría si no se hubiese relajado la restricción R_2 o bien R_5 ?

Con el enunciado que se ha dado para el problema, queda implícito que cuando se toma un arco en la solución, el coste de este arco es el que le corresponde según el enunciado del problema. No obstante esto también se puede enunciar explícitamente como una restricción que es susceptible de ser relajada. Con lo cual se pueden añadir las siguientes restricciones

- R₈. Cada uno de los ejes incluidos en la solución tiene el coste de uno de los ejes del grafo G_n .*
- R₉. El coste de cada eje de G_n solo se puede tomar una vez.*

R₁₀. Cada eje de la solución tiene el coste que le corresponde en G_n.

De nuevo sucede que no se trata de restricciones independientes, ya que en este caso es evidente que $R_{10} \Rightarrow (R_8 \wedge R_9)$, pero el hecho de considerar las tres ofrece más posibilidades de relajación. Así una nueva relajación puede ser

Relajación 2: Se relaja la restricción R₉.

En este caso el problema relajado se resuelve tomando cualquier camino entre la ciudad final de n y la ciudad de partida A que pase una y solo una vez por cada una de las ciudades no visitadas, pero eligiendo para estos arcos los costes más pequeños de G_n . Con lo que el nuevo heurístico se calcula como

$$h_2(n) = \text{suma de los costes de los } N - k + 1 \text{ ejes de menor coste de } G_n.$$

Para el nodo $n = (\text{A C B})$ se tiene que $h_2(n) = 9 + 15 + 17 + 25 = 66$. En este caso los dos heurísticos h_1 y h_2 no son comparables ya que proceden de simplificaciones distintas del problema original.

Se puede argumentar que los heurísticos h_1 y h_2 se pueden deducir fácilmente sin necesidad de aplicar el método de relajación del problema, con lo que aparentemente el método no es de gran utilidad. Sin embargo, esto no es así, ya que al haber muchas posibilidades de simplificación, hay más oportunidades de encontrar heurísticos, muchos de los cuales no serían fáciles de descubrir de otra manera. Por ejemplo si se considera una tercera relajación.

Relajación 3: Se elimina solamente de R₇.

En este caso hay que calcular el subconjunto de ejes de coste mínimo que sea conexo y en el que cada eje tenga el coste que le corresponde en G_n . Un subconjunto de estas características es un árbol de expansión mínimo que se puede calcular en tiempo polinomial con los conocidos algoritmos de Prim o Kruskal. Con lo que se obtiene el heurístico h_3 definido como

$$h_3(n) = \text{coste de un árbol de expansión mínimo de } G_n.$$

Este heurístico tampoco es comparable a los dos anteriores y aunque su cálculo es algo más complejo ($O(m^3)$ siendo m el número de ejes de G_n) en la práctica resulta muy eficiente. Para el nodo $n = (\text{A C B})$, la Figura 9.5 muestra el árbol de expansión mínimo para el correspondiente G_n , su coste es $h_3(n) = 17 + 9 + 32 + 15 = 73$.

Se pueden considerar otras relajaciones del problema. En algunos casos se obtienen heurísticos similares a alguno de los anteriores, y en otros casos se obtienen versiones del problema que siguen siendo problemas NP-duros, por lo que no sirven para el propósito de diseñar heurísticos.



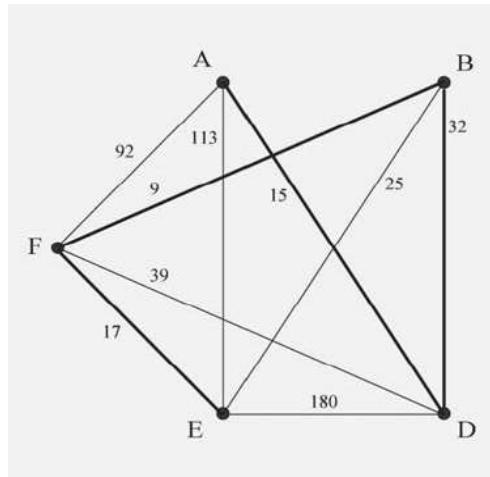


Figura 9.5: Árbol de expansión mínimo para el grafo de la Figura 8.8.

9.3. El problema de secuenciamiento de una máquina. El problema que representa un estado intermedio en el secuenciamiento de una máquina con cabezas y prioridades se puede enunciar así:

Dado un estado intermedio en el que están planificadas un subconjunto de p tareas de $\{t_1, \dots, t_N\}$, en orden $(t_{k_1}, \dots, t_{k_p})$, tal que la última termina en el instante T_p ; si C es el conjunto de tareas no planificadas, se trata de asignar a cada tarea i de C un tiempo de inicio $st_i \geq T_p$ sobre la máquina m , de forma que se minimice la suma de los tiempos de ejecución ponderados por la prioridad y que se satisfagan las restricciones siguientes:

- R_1 . *Cada tarea i debe comenzar en un tiempo $st_i \geq r_i$.*
- R_2 . *La ejecución de dos tareas en la máquina no se puede solapar en el tiempo.*
- R_3 . *La ejecución de las tareas en la máquina no se puede interrumpir durante su tiempo de procesamiento.*
- R_4 . *A cada tarea en la solución le corresponde su prioridad.*
- R_5 . *A cada tarea en la solución le corresponde la prioridad de una de las tareas del conjunto C , y no se puede asignar la prioridad de la misma tarea a dos tareas distintas.*

Con esta forma de enunciar el problema se pueden hacer varias relajaciones distintas. En primer lugar se considera la siguiente:

Relajación 1: Se prescinde de las restricciones R_1 y R_2 .

En este caso, en el problema relajado las tareas pueden comenzar en un instante $st_i = T_p$, y además se pueden solapar sobre la máquina m , con lo cual la solución óptima se

obtiene planificando todas las tareas en el instante T_p , y el valor del correspondiente heurístico es:

$$h_1(n) = \sum_{i \in C} (T_p + d_i)p_i$$

Por ejemplo, para el estado $n = (t_2 \ t_1)$ del ejemplo mostrado en la Figura 8.14 del capítulo 8, el valor de este heurístico es $h_1(n) = 20 \times 2 + 19 \times 3 = 97$, siendo en este caso $h^*(n) = 109$.

¿Cómo será el heurístico resultante de relajar solamente R_2 ?

Relajación 2: Se prescinde de R_3 y de R_4 .

En este caso cada tarea tiene que comenzar a ejecutarse en un instante st_i , tal que $st_i \geq \max(r_i, T_p)$. Las tareas no se pueden solapar, pero la ejecución de una tarea en la máquina se puede interrumpir antes de que termine su tiempo de procesamiento y asignar la máquina más tarde a esa tarea para que se procese durante el tiempo restante. Además, se pueden intercambiar las prioridades de las tareas. En este caso la solución óptima del problema relajado se puede obtener con el Algoritmo 9.9, denominado algoritmo *PS* (Preemptive Schedule). Este algoritmo requiere una prueba formal de su corrección. La idea de esta prueba se basa en que al planificar las tareas con interrupciones, dando siempre prioridad a la tarea disponible con menor tiempo de procesamiento restante, la suma de las duraciones de las tareas es mínima. Luego, si asociamos la prioridad más alta a la tarea que antes termina, y así sucesivamente, la suma de los productos por las prioridades es mínima.

En este algoritmo, ct_i se refiere al instante de finalización del último intervalo de la tarea i . Así, el heurístico resultante se calcularía como

$$h_2(n) = \text{Resultado de aplicar el Algoritmo } PS \text{ (9.9) al estado } n.$$

Para el nodo $n = (t_2 \ t_1)$, la planificación de las tareas restantes, t_3 y t_4 , se muestra en la Figura 9.6; como la mayor prioridad de las tareas no planificadas se asigna a la tarea t_4 por terminar la primera, el valor de este heurístico es $h_2(n) = 19 \times 3 + 25 \times 2 = 107$ que, como vemos, se aproxima más que el anterior al valor de $h^*(n)$.

El Algoritmo 9.9 se puede mejorar haciendo que la variable τ tome solamente los valores definidos por las cabezas y por los tiempos de finalización de las tareas, en lugar de tomar valores con incrementos de una unidad. De este modo, se puede implementar con una complejidad del orden de $O(k \log k)$ siendo k el número de tareas no planificadas en el estado n .

Al igual que en el caso del heurístico h_3 definido anteriormente para el TSP, éste es un heurístico no trivial, al que sería difícil llegar mediante un método no sistemático. Este problema tiene más posibilidades de simplificación, en particular algunas asociadas a la relajación de las duraciones de las tareas.

■



Figura 9.6: Planificación óptima de las tareas t_3 y t_4 eliminando las restricciones R_3 y R_4 .

Algoritmo 9.9 Algoritmo *PS* para el cálculo de la solución óptima del problema de secuenciamiento de tareas con prioridades relajando las restricciones R_3 y R_4 .

```

para cada instante  $\tau$  desde  $T_p$  hasta que todas las tareas de  $C$  hayan terminado hacer
    Asigna la maquina  $m$  a la tarea  $t_i$  de  $C$  con menor tiempo de procesamiento restante y tal que
     $\tau \geq r_i$ , durante el intervalo  $[\tau, \tau + 1[$ ;
fin para
Ordena las tareas de  $C$  según el tiempo  $ct_i$  de finalización, de menor a mayor;
Reasigna las prioridades de las tareas de  $C$  de modo que la mayor prioridad se asigne a la tarea
con menor tiempo de finalización, y así sucesivamente;
devolver  $\sum_{i \in C} (ct_i p_i)$ 
```

9.10 Ejercicios propuestos

9.1. Probar que si A* utiliza un heurístico monótono, entonces la secuencia de valores de $f(n)$ para los nodos expandidos es no decreciente.

9.2. Probar que si h_1 y h_2 son dos heurísticos monótonos, entonces el heurístico h definido como $h(n) = \max(h_1(n), h_2(n))$, para todo n , también es monótono.

9.3. Probar que el algoritmo de ramificación y poda de la sección 9.6 devuelve una cota inferior de la instancia del problema cuando termina sin encontrar una solución óptima.

9.4. Definir heurísticos, mediante el método de relajación del problema, para los siguientes problemas enunciados en el capítulo anterior: el problema de los misioneros y los caníbales, el problema de generación de planes de actuación de robots, el problema QAP y el problema LDMST.

9.5. Probar que el algoritmo IDA* es admisible si la función h que utiliza es admisible, y que si se añade un factor ε al actualizar el valor siguiente de la cota, entonces es ε -admisible.

9.6. Considérese el grafo de la Figura 9.7, tomado del texto de S. Russell y P. Norvig [Russell y Norvig, 2003], en el que se muestra un espacio de búsqueda y el valor del heurístico h para cada uno de los nodos. Los nodos contenidos en un cuadro son objetivos. En primer lugar se pide determinar si el heurístico h es admisible y monótono. A continuación, a partir de las propiedades del heurístico y de las condiciones, necesaria y suficiente, de expansión, indicar qué nodos se expandirán con seguridad si se aplica el algoritmo A*, cuáles no se expandirán con seguridad, y cuáles se pueden expandir o no dependiendo de la forma en que se resuelvan los empates en el valor de

la función f . Hacer lo mismo suponiendo que se utiliza el heurístico $h(n) = 0$, para todo n .

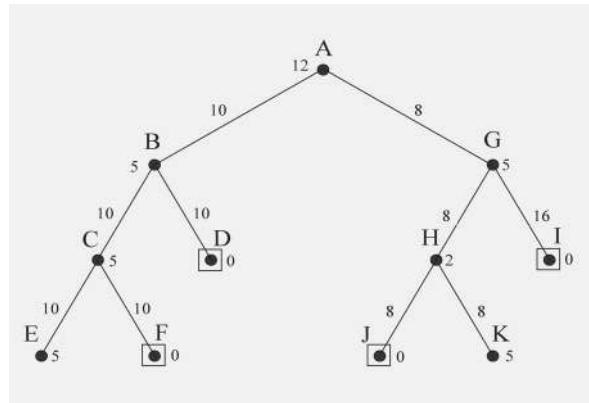


Figura 9.7: Ejemplo de árbol de búsqueda.

9.7. Aplicar los algoritmos A*, IDA* y SMA* al ejemplo de la Figura 9.7. En el caso del algoritmo SMA*, considérense al menos dos posibilidades para el tamaño de ABIERTA: 3 y 4 nodos.

9.8. Hacer lo mismo que en el ejercicio anterior, pero considerando el heurístico trivial, es decir $h(n) = 0$, para todo n .

9.9. Considérese ahora un algoritmo voraz obtenido a partir de un algoritmo A* sin más que elegir en cada expansión solamente el sucesor con menor valor de f , resolviendo los empates de forma aleatoria, ¿Cuál es el resultado de la aplicación de este algoritmo al problema de la Figura 9.7? Si en lugar de quedarse con el nodo de menor f , el algoritmo elige de forma aleatoria con una distribución de probabilidad de modo que cada sucesor tiene una probabilidad inversamente proporcional al valor de f , ¿cuál es la probabilidad de que el algoritmo voraz encuentre la solución óptima?

Referencias

- ARTIGUES, C. y FEILLET, D.: «A branch and bound method for the job-shop problem with sequence dependent setup times». *Annals of Operations Research*, 2007, **to appear**.
- BRUCKER, P.: *Scheduling Algorithms*. Springer, 4th^a edición, 2004.
- DECHTER, R. y PEARL, J.: «Generalized best first search strategies and the optimality of A*». *Journal of the Association for Computing Machinery*, 1985, **32 (3)**, pp. 505–536.
- FERNÁNDEZ, S.; GONZÁLEZ, J. y MIRA, J.: *Problemas resueltos de Inteligencia Artificial*. Addison Wesley, 1998.
- KORF, R.: «Depth-First Iterative Deepening: An Optimal Admissible Tree Search Algorithm». *Artificial Intelligence*, 1985, **27**, pp. 97–109.
- MESEGUR, P. y WALSH, T.: «Interleaved and Discrepancy Based Search». En: *Proceedings of the 13th European Conference on Artificial Intelligence, ECAI-98*, pp. 239–243, 1998.
- MICHALEWICZ, Z. y FOGEL, D. B.: *How to Solve It: Modern Heuristics*. Springer, 2000.
- NILSSON, N. J.: *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.
- NILSSON, N. J.: *Principios de Inteligencia Artificial*. Díaz de Santos, Madrid, 1987.
- NILSSON, N. J.: *Artificial Intelligence. A New Synthesis*. Morgan Kaufmann, San Francisco, California, 1998.
- NILSSON, N. J.: *Inteligencia Artificial. Una Nueva Síntesis*. McGraw Hill, Madrid, 2001.
- PEARL, J.: *Heuristics. Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, Massachusetts, 1984.
- POHL, I.: «The avoidance of relative catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving». En: *IJCAI 3*, pp. 20–23, 1973.
- RUSSELL, S.: «Efficient memory-bounded search methods». En: *10th European Conference on Artificial Intelligence (ECAI 92)*, pp. 1–5, 1992.
- RUSSELL, S. y NORVIG, P.: *Artificial Intelligence. A Modern Approach*. Prentice Hall, US, 2^a edición, 2003.

Capítulo 10

Problemas de satisfacción de restricciones (CSP)

Federico Baber Sanchís y Miguel Ángel Salido Gregorio
Universidad Politécnica de Valencia

10.1 Introducción

La programación por restricciones es una metodología software utilizada para la descripción y posterior resolución efectiva de cierto tipo de problemas, típicamente combinatorios y de optimización. Estos problemas aparecen en muy diversas áreas, incluyendo IA, investigación operativa, bases de datos y sistemas de recuperación de la información, etc., con aplicaciones en scheduling, planificación, razonamiento temporal, diseño en la ingeniería, problemas de empaquetamiento, criptografía, diagnóstico, toma de decisiones, etc. Estos problemas pueden modelarse como problemas de satisfacción de restricciones (*Constraint Satisfaction Problems* - CSP) y resolverse usando técnicas de satisfacción de restricciones. En general, se trata de grandes y complejos problemas, normalmente de complejidad NP. Las etapas básicas para la resolución de un problema CSP son su modelización y su posterior resolución mediante la aplicación de técnicas CSP específicas, que incluyen procesos de búsqueda apoyados con métodos heurísticos y procesos inferenciales. En este capítulo se presentan los conceptos, algoritmos y técnicas más relevantes en el área de los CSP, junto con diversos ejemplos y ejercicios.

Muchas decisiones que tomamos a la hora de resolver diversos problemas cotidianos están sujetas a restricciones. Decisiones tan cotidianas como fijar una cita, planificar un viaje, comprar un coche o preparar un plato de cocina puede depender de muchos aspectos interdependientes e incluso conflictivos, cada uno de los cuales está sujeto a un conjunto de restricciones que se deben satisfacer para que la decisión sea válida. Además, cuando se encuentra una solución que satisface plenamente a unos criterios, puede que no sea tan apropiada para otros, por lo que, para obtener una solución optimizada, no suele ser suficiente con obtener una única solución. Los primeros trabajos relacionados con la programación de restricciones datan de los

años 60 y 70 en el campo de la IA. Durante los últimos años, la programación de restricciones ha generado una creciente expectación entre expertos de muchas áreas debido a su potencial para la resolución de grandes y complejos problemas reales. Sin embargo, al mismo tiempo, se considera como una de las tecnologías menos conocida y comprendida. La programación de restricciones se define como el estudio de sistemas computacionales basados en restricciones. La idea de la programación de restricciones es resolver problemas mediante la declaración de restricciones sobre el dominio del problema y consecuentemente encontrar soluciones a instancias de los problemas de dicho dominio que satisfagan todas las restricciones y, en su caso, optimicen unos criterios determinados.

“Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it” (E. Freuder).

“Constraint Programming: A simple but powerful idea” (R. Dechter)

10.2 Definiciones y conceptos básicos

La programación de restricciones puede dividirse en dos ramas claramente diferenciadas: la “satisfacción de restricciones” y la “resolución de restricciones”. Ambas comparten la misma terminología, pero sus orígenes y técnicas de resolución son diferentes. La satisfacción de restricciones trata con problemas que tienen dominios finitos, mientras que la resolución de restricciones está orientada principalmente a problemas sobre dominios infinitos o dominios más complejos. En este capítulo, se tratarán principalmente los problemas de satisfacción de restricciones (CSP). Los conceptos clave en esta metodología corresponden a los aspectos de:

- La modelización del problema, que permite representar un problema mediante un conjunto finito de variables, un dominio de valores finito para cada variable y un conjunto de restricciones que acotan las combinaciones válidas de valores que las variables pueden tomar. En la modelización CSP, es fundamental la capacidad expresiva, a fin de poder captar todos los aspectos significativos del problema a modelar.
- Técnicas inferenciales que permiten deducir nueva información sobre el problema a partir de la información explícitamente representada. Estas técnicas también permiten acotar y hacer más eficiente el proceso de búsqueda de soluciones.
- Técnicas de búsqueda de la solución, apoyadas generalmente por criterios heurísticos, bien dependientes o independientes del dominio. El objetivo es encontrar un valor para cada variable del problema de manera que se satisfagan todas las restricciones del problema. En general, la obtención de soluciones en un CSP es NP-completo, mientras que la obtención de soluciones optimizadas es NP-duro, no existiendo forma de verificar la optimalidad de la solución en tiempo polinomial. Por ello, se requiere una gran eficiencia en los procesos de búsqueda.

10.2.1 Definición de un problema de satisfacción de restricciones

Un problema de satisfacción de restricciones puede ser representado mediante una terna (X, D, C) donde:

- X es un conjunto de n variables $\{x_1, \dots, x_n\}$.
- $D = < D_1, \dots, D_n >$ es un tupla de dominios finitos donde se interpretan las variables X , tal que la i -ésima componente D_i es el dominio que contiene los posibles valores que pueden asignarse a la variable x_i . La cardinalidad de cada dominio es $d_i = |D_i|$.
- $C = \{c_1, c_2, \dots, c_p\}$ es un conjunto finito de restricciones. Cada restricción k -aria c_i está definida sobre un conjunto de k variables $var(c_i) \subseteq X$, denominado su ámbito, y restringe los valores que dichas variables pueden simultáneamente tomar. Particularmente, una restricción es binaria cuando relaciona únicamente a dos variables $x_i y x_j$, y se suele denotar como c_{ij} . Todas las restricciones definidas en un CSP son conjuntivas, de manera que una solución debe de satisfacer a todas ellas.

La *instanciación* de una variable es un par variable-valor (x, a) que representa la asignación del valor a a la variable x ($x = a$). La instanciación de un conjunto de variables es una tupla de pares ordenados, donde cada par ordenado (x_i, a_i) asigna el valor $\{a_i \in D_i\}$ a la variable x_i . Una tupla $((x_1, a_1), \dots, (x_i, a_i))$ es localmente consistente si satisface todas las restricciones formadas por variables $\{x_1, \dots, x_i\}$ de la tupla. Para simplificar la notación, sustituiremos la tupla $((x_1, a_1), \dots, (x_i, a_i))$ por (a_1, \dots, a_i) .

Un valor $a_i \in D_i$ es un *valor consistente* para x_i si existe al menos una solución del CSP en la cual $x_i = a_i$. El dominio mínimo de una variable x_i es el conjunto de todos los valores consistentes para la variable, es decir, quedan excluidos aquellos valores que no forman parte de ninguna solución ($\forall x_i \in X, \forall a \in D_i, x_i = a$ forma parte de una solución del CSP).

Una *solución* a un CSP es una asignación (a_1, a_2, \dots, a_n) de valores a todas sus variables, de tal manera que se satisfagan todas las restricciones del CSP. Es decir, una solución es una tupla consistente que contiene todas las variables del problema. Una solución parcial es una tupla consistente que contiene algunas de las variables del problema. Un CSP es *consistente*, si tiene al menos una solución, es decir una tupla consistente. Dos CSPs son *equivalentes* si ambos representan el mismo conjunto de soluciones.

Ejemplo 10.1. Para el problema de las 4-Reinas, existen solamente dos soluciones (que se detallan en la Figura 10.1). Asumiendo que las variables $\{x_1, x_2, x_3, x_4\}$ representan las columnas, y sus dominios son las posibles filas $\{1, 4\}$ donde colocar las reinas, tenemos que: $(x_1, 3)$ es una instancia de la variable x_1 , y $((x_1, 2), (x_2, 4), (x_3, 1), (x_4, 3))$ es una solución del CSP, por lo que el CSP es consistente. El valor 3 es un valor consistente para x_1 , pero el valor 1 no lo es. El dominio mínimo de x_2 es $\{1, 4\}$.

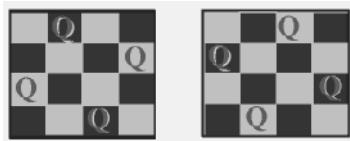


Figura 10.1: Dos soluciones al problema de las cuatro reinas.

Una vez modelado el problema como un CSP, los objetivos del CSP consisten en la *satisfacibilidad* del mismo, es decir, obtener una o varias soluciones, sin preferencia alguna, o bien obtener una solución óptima, o al menos una buena solución, en base a una función objetivo previamente definida en términos de algunas o todas las variables. En este caso particular, se trata de un Problema de Satisfacción y Optimización de Restricciones, o por su acrónimo en inglés, CSOP (*Constraint Satisfaction and Optimization Problem*).

10.2.2 Definición y tipología de las restricciones

Las restricciones se caracterizan fundamentalmente por su *aridad*, que es el número de variables involucradas en dicha restricción. Una restricción unaria es una restricción sobre una sola variable. Una restricción binaria es una restricción que consta de dos variables. Una restricción no binaria (o *n*-aria) es una restricción que involucra a un número arbitrario de 3 o más variables.

Ejemplo 10.2. La restricción $x \leq 5$ es una restricción unaria sobre la variable x . La restricción $x_4 - x_3 \neq 3$ es una restricción binaria. La restricción $2x_1 - x_2 + 4x_3 \geq 4$ es una restricción ternaria. Por último, un ejemplo de restricción *n*-aria sería $x_1 + 2x_2 - x_3 + 5x_4 \leq 9$.

Una restricción sobre un conjunto de variables puede definirse *extensionalmente* mediante un conjunto de tuplas válidas o no válidas y también *intensionalmente* mediante una función aritmética. La representación extensional de una restricción *k*-aria está formada por un conjunto de tuplas, cada una con *k* elementos, y expresa el conjunto de valores que las *k* variables pueden tomar simultáneamente. Claramente, en el caso de CSP continuos es imposible representar las restricciones extensionalmente ya que generalmente hay un número infinito de tuplas válidas.

Ejemplo 10.3. Consideremos una restricción que involucra a las variables x_1, x_2, x_3, x_4 , con dominios $\{1, 2\}$, donde la suma entre las variables x_1 y x_2 es menor o igual que la suma entre x_3 y x_4 . Esta restricción puede representarse intensionalmente mediante la expresión $x_1 + x_2 \leq x_3 + x_4$. También, podría representarse extensionalmente mediante el conjunto de tuplas permitidas $\{(1, 1, 1, 1), (1, 1, 1, 2), (1, 1, 2, 1), (1, 1, 2, 2), (2, 1, 2, 2), (1, 2, 2, 2), (1, 2, 1, 2), (1, 2, 2, 1), (2, 1, 1, 2), (2, 1, 2, 1), (2, 2, 2, 2)\}$.

En el caso de una definición intensional, podemos tener restricciones *no-disyuntivas* o *disyuntivas*, según expresen una única relación o más de una relación disyuntiva entre las variables. Por ejemplo, asumiendo un modelo con las relaciones elementales

$\{<, =, >\}$, $(x_1 < x_2)$ es una restricción no-disyuntiva, mientras que $(x_1 < x_2 \vee x_1 > x_2)$, es decir, $x_1 \neq x_2$, sería una restricción disyuntiva. Por otra parte, las restricciones también pueden ser *cualitativas*, cuando expresan una relación de orden entre las variables (por ejemplo, $x_1 < x_2$), o *métricas* cuando expresan una distancia métrica entre las mismas (por ejemplo, $x_1 < x_2 + 7$). Las restricciones métricas requieren una métrica en el dominio de interpretación de las variables.

Un tipo especial de restricciones son las restricciones lineales. Una *relación lineal* sobre $X = \{x_1, \dots, x_k\}$ es una expresión de la forma:

$$\sum_{i=1}^k p_i x_i \{<, \leq, \neq, \geq, >\} b$$

donde p_i son los coeficientes y $b \in \mathbb{R}$. En base a combinaciones lógicas de desigualdad (\neq) e igualdad (\leq), se pueden expresar todas las relaciones en $\{<, \leq, \neq, \geq, >\}$. Las *Restricciones Lineales Disyuntivas (DLR)* son disyunciones de restricciones lineales.

10.3 Ejemplos de CSP y su modelización

El primer paso en la resolución de un CSP es su modelización, es decir, su representación en términos de variables, dominios y restricciones. Al igual que ocurre con el lenguaje natural, la modelización de un problema se puede realizar de muchas maneras diferentes. Respecto a la modelización de un problema CSP hay dos aspectos básicos:

- La potencia expresiva de las restricciones, es decir, la capacidad de modelar las restricciones realmente existentes en el problema real.
- La eficiencia de la representación, ya que dependiendo de la modelización CSP, el problema se resolverá con más o menos eficiencia.

10.3.1 Coloración del mapa

Este problema parte de un conjunto de colores posibles para colorear cada región del mapa, de manera que regiones adyacentes tengan distintos colores. En la formulación del CSP, definimos una variable por cada región del mapa, y el dominio de cada variable es el conjunto de colores disponible. Para cada par de regiones contiguas existe una restricción sobre las variables correspondientes que no permite la asignación de idénticos valores a las variables. En el caso de la Figura 10.2, tenemos un mapa con cuatro regiones x, y, z, w para ser coloreadas con los posibles colores Rojo, Verde, Azul. La formulación CSP sería:

- Variables: $\{x, y, z, w\}$.
- Dominio: $\{\text{Rojo}, \text{Verde}, \text{Azul}\}$, único para las tres variables.
- Restricciones (definición intensional): $\{x \neq y, x \neq w, x \neq z, y \neq w, w \neq z\}$.

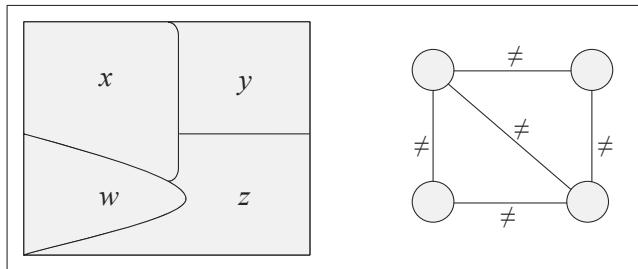


Figura 10.2: El problema de coloración del mapa.

Un CSP binario, puede ser representado mediante una red de restricciones, donde los nodos representan las variables y los arcos representan las restricciones entre las mismas. En la Figura 10.2-derecha, se representa la red correspondiente al problema del ejemplo, donde las variables correspondientes a regiones adyacentes están conectadas por una arista. Hay cinco restricciones en el problema, es decir, cinco aristas en la red. Una solución para el problema es la asignación $(x, r), (y, v), (z, v), (w, a)$. En esta asignación, todas las variables adyacentes tienen valores diferentes.

10.3.2 Criptografía

El típico problema criptográfico "*send + more = money*" consiste en asignar a cada letra $\{s, e, n, d, m, o, r, y\}$ un dígito diferente del conjunto $\{0, \dots, 9\}$ de forma que se satisfaga: $send + more = money$.

$$\begin{array}{r}
 & s & e & n & d \\
 + & m & o & r & e \\
 \hline
 m & o & n & e & y
 \end{array}$$

La manera más fácil de modelar este problema es asignando una variable a cada una de las letras, todas ellas con un dominio $\{0, \dots, 9\}$ y con las restricciones que obliguen a que todas las variables tomen valores distintos y con la correspondiente restricción para que se satisfaga "*send + more = money*". De esta forma las restricciones son:

- $10^3(s + m) + 10^2(e + o) + 10(n + r) + d + e = 10^4m + 10^3o + 10^2n + 10e + y$.
- restricción de todas diferentes: $\neq(s, e, n, d, m, o, r, y)$.

La restricción de "todas diferentes" puede reemplazarse por un conjunto de restricciones binarias, $\{s \neq e, s \neq n, \dots, r \neq y\}$, donde $(x_i \neq x_j)$ es la relación disyuntiva $x_i \{<, >\} x_j$. Sin embargo, para el algoritmo de resolución más general como es Backtracking (que veremos en las siguientes secciones), este modelo no es muy eficiente porque todas las variables necesitan ser instanciadas antes de comprobar estas dos restricciones. De esta manera, no se podría podar el espacio de búsqueda durante

el propio proceso a fin de agilizar la búsqueda. Además, la primera restricción es una igualdad en la que forman parte todas las variables del problema (restricción global) por lo que dificulta el proceso de consistencia. Un modelo más eficiente podría utilizar los bits de acarreo para descomponer la ecuación anterior en un conjunto de pequeñas restricciones. Esto requeriría incluir tres variables *portadoras* adicionales, c_1, c_2, c_3 , cuyo dominio es $\{0, 1\}$:

- $e + d = y + 10c_1;$
- $c_1 + n + r = e + 10c_2;$
- $c_2 + e + o = n + 10c_3;$
- $c_3 + s + m = 10m + o$
- restricción de todas diferentes: $\neq (s, e, n, d, m, o, r, y).$

La ventaja de este modelo es que estas restricciones con menor aridad, puedan comprobarse antes en la búsqueda de backtracking, y así podarse muchas inconsistencias.

10.3.3 El problema de las N -reinas

Este problema consiste en colocar N -reinas en un tablero de ajedrez de dimensión $N \times N$, de forma que ninguna reina esté amenazada. De esta forma no puede haber dos reinas en la misma fila, misma columna, o misma diagonal. Si asociamos cada columna a una variable y su valor representa la fila donde se coloca una reina, el problema puede ser formulado como:

- Variables: $\{x_i\}, i = 1..N.$
- Dominio = $\{1, 2, 3, \dots, N\}$, para todas las variables.
- Restricciones: $(\forall x_i, x_j, i \neq j):$
 - $x_i \neq x_j$, No en la misma fila.
 - $x_i - x_j \neq i - j$, No en la misma diagonal SE.
 - $x_j - x_i \neq i - j$, No en la misma diagonal SO.

Resultando para el caso particular de $N=4$ (véase la Figura 10.3):

- $|x_1 - x_2| \neq 1 ;$
- $|x_1 - x_3| \neq 2 ;$
- $|x_1 - x_4| \neq 3 ;$
- $|x_2 - x_3| \neq 1 ;$

- $|x_2 - x_4| \neq 2$;
- $|x_3 - x_4| \neq 1$;
- $\neq (x_1, x_2, x_3, x_4)$;

	X1	X2	X3	X4
1				
2				
3				
4				

Figura 10.3: Tablero para modelado del problema de las 4-reinas

En los ejercicios propuestos al final del capítulo se muestran otros ejemplos sobre la idoneidad de una buena especificación del CSP, tanto respecto a la simplicidad de la especificación, como con respecto a la eficiencia en el proceso de búsqueda.

10.4 Técnicas CSP

Las técnicas más usuales que se llevan a cabo para manejar un CSP se pueden agrupar en tres tipos: búsqueda sistemática, técnicas inferenciales y técnicas híbridas.

10.4.1 Métodos de búsqueda

Los métodos de búsqueda se centran en explorar el espacio de estados del problema. Estos métodos pueden ser *completos*, explorando todo el espacio de estados en busca de una solución, o *incompletos* si solamente exploran una parte del espacio de estados. Los métodos que exploran todo el espacio de búsqueda garantizan encontrar una solución, si existe, o demuestran que el problema no es resoluble. La desventaja de estos algoritmos es que son muy costosos. Los dos métodos completos más usuales son:

- **Generar y Testear (GT):** Este método genera las posibles tuplas de instanciación de todas las variables de forma sistemática y después testea sucesivamente sobre cada instanciación si se satisfacen todas las restricciones del problema. La primera combinación que satisface todas las restricciones, será la solución al problema. Mediante este procedimiento, el número de combinaciones generado por este método es el Producto Cartesiano de la cardinalidad de los dominios de las variables: $\prod_{i=1,n} d_i$. Esto es un inconveniente, ya que se realizan muchas instanciações erróneas de valores a variables que después son rechazadas en la

fase de testeo. Por ejemplo, para el caso del problema de las 4-reinas, se generarían $4^4 = 256$ tuplas a testear. El proceso de generación requeriría $256 \times 4 = 1024$ asignaciones de variable.

- **Backtracking Cronológico (BT):** Este método realiza una exploración en profundidad del espacio de búsqueda, instanciando sucesivamente las variables y comprobando ante cada nueva instancia si las instancias parciales ya realizadas son localmente consistentes. Si es así, sigue con la instancia de una nueva variable. En caso de conflicto, intenta asignar un nuevo valor a la última variable instanciada, si es posible, y en caso contrario retrocede a la variable asignada inmediatamente anterior.

En el Algoritmo 10.1 se puede ver el típico proceso de BT aplicado a la resolución de CSP, donde $V[n]$ representa el vector de asignaciones a las variables (x_1, x_2, \dots, x_n) del problema, ordenadas según un determinado criterio. La función $Comprobar(K, V[n])$ comprueba la validez de las k instancias ya realizadas $\{V[1], V[2], \dots, V[k]\}$, es decir, si se satisfacen las restricciones locales existentes entre las variables (x_1, x_2, \dots, x_k) , como se indica en la Expresión 10.1.

$$Comprobar(k, V[n]) = Tsii : \forall i (1 \leq k) : (V[k], V[i]) \in c_{ki} \quad (10.1)$$

Si el algoritmo finaliza con éxito, el vector $V[n]$ representa la tupla de valores solución. La Figura 10.4 muestra una sencilla aplicación del algoritmo, donde se indican los puntos de backtracking y la solución encontrada. Los algoritmos tipo BT realizan sólo una comprobación *hacia atrás* en cada instancia, por lo que pertenece al tipo de algoritmos *Look-Backward* (véase la sección 10.4.3.1).

Algoritmo 10.1 Algoritmo de backtracking cronológico.

```

procedimiento Backtracking( $k, V[n]$ ) ; Llamada inicial: Backtracking(1,  $V[n]$ )
  inicio
     $V[k] = Selección(d_k)$  ; Selecciona un valor de  $d_k$  para asignar a  $x_k$ 
    si Comprobar( $k, V[n]$ ) entonces
      si  $k = n$  entonces
        devolver  $V[n]$  ; Es una solución
      si no
        Backtracking( $k + 1, V[n]$ )
      fin si
    si no
      si quedan_valores( $d_k$ ) entonces
        Backtracking( $k, V[n]$ )
      si no
        si  $k = 1$  entonces
          devolver  $\emptyset$  ; Fallo
        si no
          Backtracking( $k - 1, V[n]$ )
        fin si
      fin si
    fin si
  fin Backtracking

```

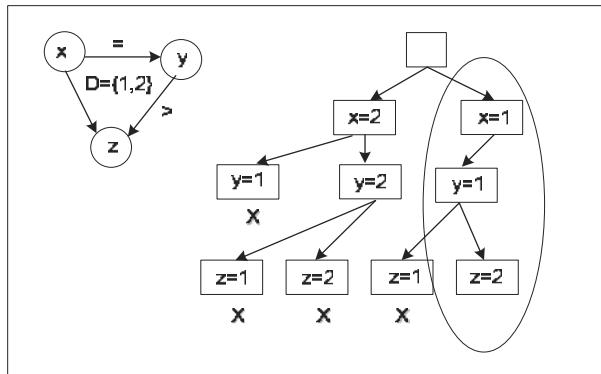


Figura 10.4: Aplicación del Backtracking Cronológico.

La Figura 10.5 muestra la aplicación del algoritmo BT al problema de las 4-reinas, con una ordenación lexicográfica de variables y valores. Los nodos marcados con X representan que la asignación realizada no es consistente con las asignaciones previas, y por tanto son puntos de backtracking. Como cabía esperar, BT no realiza tanta búsqueda como GT, al detectar cada inconsistencia en cuanto aparece. Pero para este simple problema, aún requiere generar 26 asignaciones.

El backtracking cronológico es un algoritmo muy simple pero también es muy ineficiente. El problema es que tiene una visión local del problema, de forma que sólo comprueba las restricciones entre las variables ya instanciadas, e ignora la relación entre dichas variables (x_1, x_2, \dots, x_k) y las que quedan por instanciar (x_{k+1}, \dots, x_n). La mejora del algoritmo incluye dos modificaciones fundamentales:

- Mejorar la función *Comprobar*, tal que también compruebe la validez de la instanciación parcial efectuada respecto a las variables pendientes de asignar. Esto permitiría detectar cuento antes si la solución parcial efectuada de las k variables forma o no parte de una solución global, evitando profundizar en una secuencia de asignaciones que no conduzca a una solución. Esto se realiza mediante un proceso de propagación de las instanciaciones parciales ya realizadas al resto de la red tal y como veremos en la sección 10.4.3.
- Establecer un orden de instanciación de las variables (orden del vector $V[n]$) o establecer un criterio para la *selección* de los valores de los dominios correspondientes ($Selección(d_k)$). Estos métodos para la ordenación de variables y valores se verán en la sección 10.5.

10.4.2 Técnicas de inferencia

Los procesos inferenciales en un CSP tienen como objetivo deducir nuevas restricciones, derivadas de las explícitamente conocidas sobre el problema. Concretamente, estas técnicas borran valores inconsistentes de los dominios de las variables o inducen

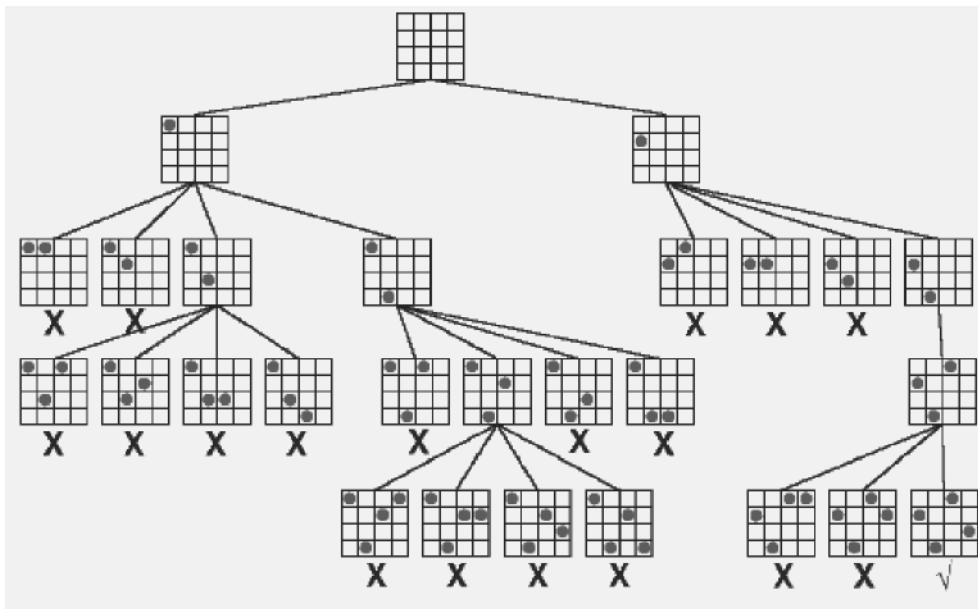


Figura 10.5: Backtracking Cronológico aplicado al problema de las 4-reinas.

restricciones implícitas entre las variables, obteniendo un nuevo CSP, equivalente al inicial, donde se han hecho explícitas las nuevas restricciones implícitamente contenidas en el primero. La obtención de estas nuevas restricciones permite:

1. Obtener respuestas a preguntas sobre el problema, relativas a restricciones implícitamente existentes entre las variables o sobre sus dominios.
2. Acotar el espacio de soluciones, al haberse eliminado valores inconsistentes, haciendo más eficientes los procesos de búsqueda.

Los algoritmos de búsqueda sistemática para la resolución de CSP vistos en la sección 10.4.1 tienen como base la búsqueda basada en backtracking. Sin embargo, esta búsqueda sufre con frecuencia una explosión combinatoria en el espacio de búsqueda, y por lo tanto no es por sí solo un método suficientemente eficiente para resolver CSP. Una de las principales dificultades es la aparición de inconsistencias locales que van apareciendo continuamente [Mackworth, 1977]. Las inconsistencias locales son valores individuales o combinación de valores de las variables que no pueden participar en la solución. Por ejemplo, si el valor a de la variable x es incompatible con todos los valores de una variable y pendiente de asignación, ligada a x mediante una restricción, entonces a es inconsistente y no formará parte de ninguna solución del problema. Por lo tanto, si forzamos alguna propiedad de *consistencia local* podemos borrar todos los valores que son inconsistentes con respecto a dicha propiedad. Esto no significa que todos los valores que no puedan participar en una solución sean borrados.

Puede haber valores que son consistentes con respecto a un cierto nivel de consistencia local, pero son inconsistentes con respecto a cualquier otro nivel mayor de consistencia local. Así, *consistencia global* asegura que todos los valores de los dominios de las variables que no pueden participar en una solución son eliminados.

Las restricciones explícitas en un CSP, que generalmente coinciden con las que se conocen explícitamente del problema a resolver, generan cuando se combinan, restricciones implícitas que pueden causar inconsistencias locales. Si un algoritmo de búsqueda no almacena las restricciones implícitas, repetidamente redescubrirá la inconsistencia local causada por ellas y malgastará esfuerzo de búsqueda tratando repetidamente de intentar instanciaciones que ya han sido probadas. Veamos el siguiente ejemplo.

Ejemplo 10.4. Tenemos un problema con tres variables x, y, z , con los dominios $\{0, 1\}, \{2, 3\}$ y $\{1, 2\}$ respectivamente. Hay dos restricciones en el problema: $y < z$, $x \neq y$. Si asumimos que la búsqueda mediante backtracking trata de instanciar las variables en el orden x, y, z , entonces probará todas las posibles 2^3 combinaciones de valores para las variables antes de descubrir que no existe solución alguna. Si miramos la restricción entre la variable y y la variable z podremos ver que no hay ninguna combinación de valores para las dos variables que satisfagan la restricción. Si el algoritmo pudiera identificar esta inconsistencia local antes, se evitaría un gran esfuerzo de búsqueda.

Los procesos inferenciales están ligados al nivel de consistencia. Un proceso inferencial completo deduciría toda la información contenida en el CSP y va ligado a un nivel de consistencia global. Sin embargo, tiene en general un coste exponencial. Los procesos inferenciales incompletos deducen sólo parte de la información contenida en el CSP y van ligados a niveles de consistencia local. En general, su coste es de tipo polinómico, por lo que resultan más interesantes y aplicables. Veamos a continuación distintos niveles de consistencia en un CSP.

10.4.2.1 Consistencia de nodo (1-consistencia)

La consistencia local más simple de todas es la *consistencia de nodo* o *nodo-consistencia*. Una variable x_i es nodo-consistente si y solo si todos los valores de su dominio D_i son consistentes con las restricciones unarias sobre la variable. Un CSP es nodo-consistente si y solo si todas sus variables son nodo-consistentes:

$$\forall x_i \in X, \forall c_i \in C, \exists a \in D_i : (a) \in c_i$$

Por ejemplo, en la Figura 10.6 consideramos una variable x en un problema con dominio $[2, 15]$ y la restricción unaria $x \leq 7$. La consistencia de nodo eliminará el intervalo $[8, 15]$ del dominio de x . El algoritmo que garantiza un CSP nodo-consistente es trivial (elimina los valores inconsistentes de los dominios de las variables respecto a sus restricciones unarias) y tiene un coste lineal respecto al tamaño del problema, habitualmente asociado al número de variables n .

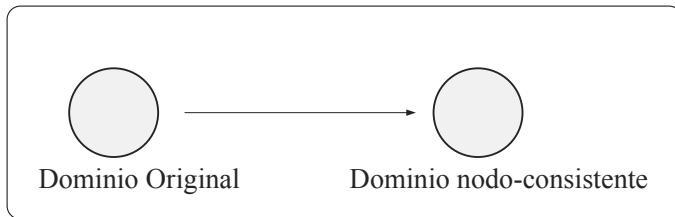


Figura 10.6: Consistencia de nodo.

10.4.2.2 Consistencia de arco (2-consistencia)

Dos variables (x_i, x_j) son arco-consistentes si para cada valor a en D_i hay al menos un valor b en D_j tal que se satisface la restricción existente entre x_i, x_j . Debe tenerse en cuenta que una restricción c_{ij} implica una restricción simétrica c'_{ji} . Un CSP es *arco-consistente* si cada par de variables del CSP es arco-consistente:

$$\forall c_{ij} \in C, \forall a \in D_i, \exists b \in D_j : (a, b) \in c_{ij}$$

Cualquier valor en el dominio D_i de la variable x_i que no es arco-consistente es eliminado de D_i , ya que no puede formar parte de solución alguna. Dicho de otra manera, una solución parcial de una variable del problema puede ser extendida a una solución parcial con cualquier otra variable. Los algoritmos de arco-consistencia recorren todos los arcos de la red, por lo que tienen un coste cuadrático respecto al número de variables n . El Algoritmo 10.2 describe un algoritmo eficiente que garantiza la arco-consistencia de un CSP [Mackworth, 1977], acotando los dominios de las variables para que se cumpla dicha propiedad.

En el ejemplo de la Figura 10.7, podemos observar que el problema es arco-consistente, ya que para cada valor $a \in [3, 6]$ hay al menos un valor $b \in [8, 10]$ de manera que se satisface la restricción $x_i < x_j$. Sin embargo, si la restricción fuese $x_i = x_j$ no sería arco-consistente.

Los procesos de k -consistencia suelen aplicarse antes de los procesos de búsqueda de la solución a fin de acotar los dominios y restricciones de la red, restringiendo con ello el espacio de búsqueda. Por ejemplo, en la Figura 10.8, que corresponde a un caso particular del coloreado de mapas, podemos ver que el proceso de arco-consistencia elimina suficientes valores de los dominios de las variables tal que la solución es directamente obtenida sobre los valores restantes de los dominios. También puede verse el ejercicio resuelto 1 al final del capítulo, donde se muestra otro ejemplo de arco-consistencia.

Un caso particular de arco-consistencia es la *arco-consistencia dirigida*. Ello solo exige que para cualquier valor $a \in D_i$ de cualquier variable x_i , y para todos los arcos *dirigidos* c_{ij} , exista un valor b en el dominio D_j que satisfaga la restricción $(x_i c_{ij} x_j)$.

Algoritmo 10.2 Algoritmo para garantizar la arco-consistencia

Procedimiento $AC(X, n, D, C)$; Devuelve el CSP arco-consistente

inicio

$Q \leftarrow C$; queue: cola de arcos, inicialmente los arcos en csp.

mientras $Q \neq \emptyset$ **hacer**

$c_{ij} \leftarrow REMOVE_FIRST(Q)$; c_{ij} restricción entre x_i, x_j

si $Revisar(c_{ij})$ **entonces**

para todo $x_k \in NEIGHBORS[x_i]$ **hacer**

 añadir c_{ki} a Q

fin para

fin si

fin mientras

fin AC ;

Función $Revisar(c_{ij})$; Retorna T si y sólo si eliminamos valores de D_i

inicio

$removed \leftarrow false$;

para todo $x \in DOMAIN[x_i]$ **hacer**

si ningún_valor $y \in DOMAIN[x_j]$ satisface c_{ij} **entonces**

 eliminar x de $DOMAIN[x_i]$;

$removed \leftarrow true$

fin si;

fin para

devolver $removed$

fin $Revisar$;

10.4.2.3 Consistencia de caminos (3-consistencia)

La consistencia de caminos [Montanari, 1974] o *senda-consistencia* (en inglés, Path-Consistency), es un nivel más alto de consistencia local que la arco-consistencia. La consistencia de caminos requiere que, para cada asignación de dos variables (x_i, a) , (x_j, b) consistente con la restricción directa que existe entre ellas c_{ij} , exista también un valor para cada variable a lo largo del camino entre ellas de forma que todas las restricciones a lo largo del camino se satisfagan (véase la Figura 10.9).

Montanari demostró que un CSP satisface la consistencia de caminos si y sólo si todos los caminos de longitud 2 cumplen la consistencia de caminos. En otras palabras, si y sólo si cualquier solución local entre dos variables, es decir, consistente con la restricción entre ellas, es extensible a cualquier tercera variable:

$$\forall(a, b) \in c_{ij}, \forall x_k \in X, \exists c \in D_k, (a, c) \in c_{ik} \wedge (b, c) \in c_{jk}$$

Basándose en esta propiedad, el algoritmo PC [Mackworth, 1977; Montanari, 1974] (véase el Algoritmo 10.3) obtiene un CSP senda-consistente para CSP binarios. Más concretamente, el algoritmo acota las restricciones del CSP inicial, de forma que el CSP resultante cumple la consistencia de caminos.

El algoritmo PC (véase el Algoritmo 10.3) recorre todas las subredes de 3 nodos (conjunto Q), forzando la consistencia de senda de cada restricción c_{ij} mediante la función Revisar. En el caso de que la restricción c_{ij} resulte modificada (en su forma extensional), se vuelven a tratar todas las subredes de 3 nodos que incluyan a x_i, x_j

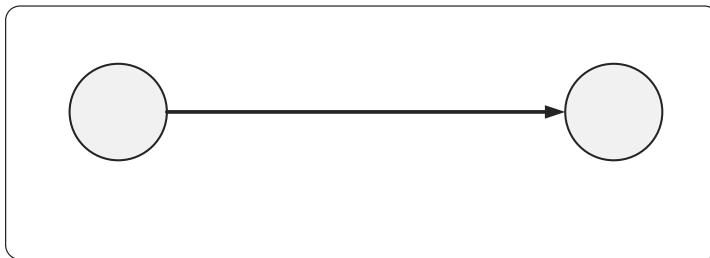


Figura 10.7: Arco-consistencia.

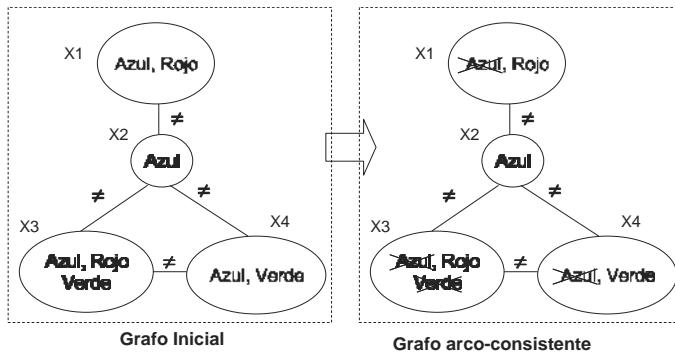


Figura 10.8: Ejemplo de proceso de Arco-Consistencia.

para comprobar si siguen localmente consistentes. De esta forma, cada subred de 3 nodos, cada senda de longitud 2, resulta localmente consistente. Por lo tanto, el CSP resultante será senda-consistente. En el caso en que al modificar la restricción c_{ij} quedase vacía, $c_{ij} = \emptyset$, sería un CSP no consistente.

Un ejemplo de senda-consistencia puede verse en la Figura 10.10, donde, a partir de la red inicial de restricciones R, el algoritmo obtiene la red senda consistente R': cualquier camino entre cualquier par de nodos lo es.

Una restricción c_i es consistente si existe una solución en la cual dicha restricción se cumpla. Una restricción c_i es *mínima* si y solo si consiste únicamente de tuplas consistentes, es decir, forman parte de alguna solución. Un CSP es *mínimo* si y solo si todas sus restricciones y dominios son mínimos. Puede verse que la red R' de la Figura 10.10 es senda-consistente, pero no es mínima, ya que no hay ninguna solución posible donde $B = C$.

10.4.2.4 *k*-consistencia. Consistencia global

En general, se dice que una red es *k-consistente* si y solo si dada cualquier instanciación de $k-1$ variables, que satisfagan todas las restricciones entre ellas, existe al menos una instanciación de una variable k , tal que se satisfacen las restricciones entre

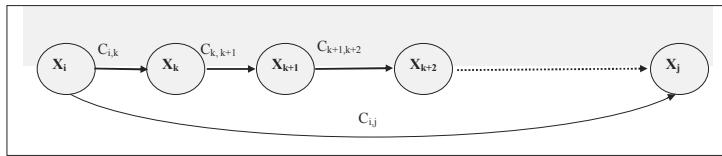


Figura 10.9: Senda-consistencia.

Algoritmo 10.3 Algoritmo de senda-consistencia.

```

procedimiento  $PC(X, n, D, C)$ 
  inicio
     $Q := \{(i, j, k) \mid 1 = i < j = n, 1 \leq k \leq n, i \neq k, j \neq k\}$  ; Todas las subredes de 3 nodos
    mientras  $Q \neq \emptyset$  hacer
      Select  $(i, j, k)$  from  $Q$ ;
      si  $Revisar(i, j, k)$  entonces
         $Q := Q \cup \{(l, i, j), (l, j, i) \mid 1 \leq l \leq n, l \neq j, l \neq i\}$ 
      fin si
    fin mientras
    fin  $PC$ ;

  función  $Revisar(i, j, k)$ 
  inicio
     $cambiado := false$ ;
    para todo  $(a, b) \in c_{ij}$  hacer
      si  $\neg \exists c \in d_k / (a, c) \in c_{ik} \wedge (c, b) \in c_{kj}$  entonces
         $c_{ij} := c_{ij} - (a, b)$ ; Acota la restricción  $c_{ij}$ , pero  $cambiado := true$ 
        ; si resulta  $c_{ij} = \{\emptyset\}$ 
      fin si ;CSP no consistente
    fin para
    devolver  $cambiado$ ;
  fin  $Revisar$ ;

```

las k variables [Freuder, 1982]. En otras palabras, cualquier solución en una subred de tamaño $k-1$, puede ser extendida a una solución en una subred de tamaño k , que contenga a la anterior. En consecuencia, cada subred de k variables es localmente consistente. Cuando una red es *i-consistente* $\forall i, i \leq k$ se dice que es *fuertemente i-consistente*.

Un CSP es *consistente* si tiene al menos una solución. Un CSP es *globalmente consistente* si es *i-consistente* $\forall i, i \leq n$, es decir, cualquier instanciación localmente consistente entre i variables ($1 \leq i \leq n$), puede formar parte de una solución global. De esta manera, un CSP *globalmente consistente* contiene solamente aquellas combinaciones de valores que forman parte de al menos una solución, siendo una representación compacta y conservadora de todas las soluciones de un CSP. Es correcto en el sentido de que la red nunca admite una combinación de valores que no desembocue en una solución. Es completo en el sentido de que todas las soluciones están representadas en él. Un CSP *globalmente consistente* es un CSP mínimo. En una red de restricciones globalmente consistente, la búsqueda puede llevarse a cabo sin necesidad de backtracking [Freuder, 1982].

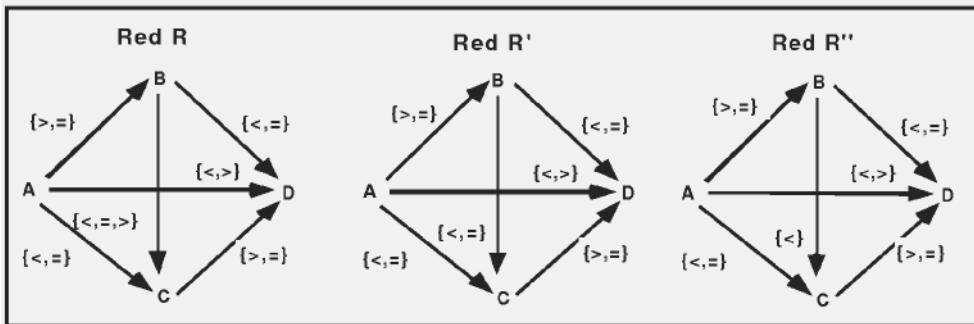


Figura 10.10: Red inicial (R), red senda-consistente (R'), red mínima (R'').

En general, un etiquetado *globalmente consistente* de una red requiere representar de forma explícita restricciones para todas las variables del problema, es decir, generar etiquetas $n-1$ -dimensionales para un problema de talla n forzando la n -consistencia. Esta tarea tiene una complejidad exponencial en el peor caso. Para clases especiales de problemas, niveles bajos de consistencia, son equivalentes a la consistencia del CSP. Estos resultados permiten a los algoritmos polinómicos llevar a cabo etiquetados consistentes. Los podemos resumir en los siguientes:

- La arco-consistencia es equivalente a la consistencia global cuando la red de restricciones es un árbol [Freuder, 1982].
- La consistencia de caminos es equivalente a la consistencia global cuando el CSP es convexo y binario [Dechter y otros, 1991; Van Beek, 1991], siendo un CSP convexo aquel cuyas restricciones determinan un espacio de soluciones convexo.

El problema para conocer si un CSP es consistente (tiene al menos una solución) se denomina *problema de satisfacibilidad*. Este problema, así como el problema equivalente de encontrar una solución, es en general NP. Existen diferentes aproximaciones para identificar *clases tratables* de CSP, las cuales corresponden a dos líneas fundamentales:

1. Identificación por la estructura de la red de restricciones, independientemente del tipo de relaciones que contenga. En esta línea resulta interesante encontrar estructuras de búsqueda similares a árboles (véase la sección 10.5.1).
2. Identificación por el tipo de relación entre variables, debido a las especiales propiedades de dichas relaciones. Particularmente, el problema de obtener la consistencia (o encontrar soluciones) en una red binaria no disyuntiva es n^3 . Asimismo, en una red binaria con las relaciones $\{<, =, >\}$, el problema de obtener la consistencia tiene un coste n^4 .

10.4.3 Técnicas híbridas

En la sección 10.4.1 se han visto técnicas de búsqueda de soluciones en un CSP que, en general, resultan exponenciales con el tamaño del problema. Por otra parte, en la sección 10.4.2 se han visto técnicas inferenciales que eliminan valores inconsistentes de los dominios de las variables o inducen nuevas restricciones implícitas entre las variables restringiendo las previas, de forma que obtienen un nuevo CSP, más restringido y equivalente al inicial, que permite acotar el espacio de búsqueda.

Por ello, las técnicas inferenciales se usan como etapas de preprocessamiento donde se detectan y se eliminan inconsistencias locales antes de empezar la búsqueda con el fin de reducir el árbol de búsqueda. Dependiendo del nivel de consistencia del preprocessamiento, se acotará más o menos espacio de búsqueda, a costa de un mayor o menor esfuerzo computacional en el proceso inferencial previo. Por otra parte, las técnicas inferenciales pueden incluirse en el propio proceso de búsqueda, dando lugar a los *algoritmos de búsqueda híbridos* que analizamos en esta sección. La complejidad exponencial para resolver un CSP dado está principalmente relacionada con el tamaño de los dominios de sus variables.

10.4.3.1 Algoritmos Look-Backward

Los algoritmos Look-Backward tratan de explotar la información del problema para comportarse más eficientemente en las situaciones sin salida. Al igual que el backtracking cronológico, los algoritmos Look-Backward llevan a cabo la comprobación de la consistencia *hacia atrás*, es decir, entre la variable actualmente instanciada y las pasadas ya instanciadas. Veamos algunas variantes de algoritmos Look-Backward.

- **Backjumping** (BJ) [Gaschnig, 1979] es un algoritmo parecido al backtracking cronológico, excepto que se comporta de una manera más inteligente cuando encuentra situaciones sin salida. En vez de retroceder a la variable anteriormente instanciada, BJ salta a la variable x_j más profunda, es decir más cerca de la variable actual, que está en conflicto con la variable actual x_i donde $j < i$. Decimos que una variable instanciada x_j está en conflicto con una variable x_i si la instanciación de x_j evita uno de los valores en x_i , debido a la restricción entre x_j y x_i . Cambiar la instanciación de x_j puede hacer posible encontrar una instanciación consistente de la variable actual.
- **Conflict-directed Backjumping** (CBJ) [Prosser, 1993] tiene un comportamiento de salto hacia atrás más sofisticado que BJ. Cada variable x_i tiene un *conjunto conflicto* formado por las variables pasadas que están en conflicto con x_i . En el momento en el que la comprobación de la consistencia entre la variable actual x_i y una variable pasada x_j falla, la variable x_j se añade al conjunto conflicto de x_i . En una situación sin salida, CBJ salta a la variable más profunda en su conjunto conflicto, por ejemplo, x_k , donde $k < i$. Al mismo tiempo se incluye el conjunto conflicto de x_i al de x_k , por lo que no se pierde ninguna información sobre conflictos. Obviamente, CBJ necesita unas estructuras de datos más complicadas para almacenar los conjuntos conflicto.

- **Learning** [Frost y Dechter, 1994] es un método que almacena las restricciones implícitas que son derivadas durante la búsqueda y las usa para podar el espacio de búsqueda. Por ejemplo, cuando se alcanza una situación sin salida en la variable x_i , entonces sabemos que la tupla de asignaciones $((x_1, a_1), \dots, (x_{i-1}, a_{i-1}))$ nos lleva a una situación sin salida. Así, nosotros podemos *aprender* que una combinación de asignaciones para las variables x_1, x_{i-1} no está permitida.

10.4.3.2 Algoritmos Look-Ahead

Como ya indicamos anteriormente, los algoritmos Look-Backward tratan de reforzar el comportamiento del backtracking mediante un comportamiento más inteligente cuando se encuentran situaciones sin salida. Sin embargo, todos ellos todavía llevan a cabo la comprobación de la consistencia solamente hacia atrás: detectan las inconsistencias en cuanto aparecen, pero ignoran las consecuencias de las instanciaciones parciales ya realizadas sobre las restricciones y dominios existentes en las futuras variables a instanciar.

Los algoritmos *Look-Ahead* hacen una comprobación *inferencial* hacia adelante en cada instanciación, integrando un proceso inferencial durante el propio proceso de búsqueda, por lo que también se denominan *técnicas híbridas*. Esto también se conoce como la *propagación* de los efectos de cada nueva instanciación al resto de la red. Ello permite: (i) acotar las restricciones y dominios de las variables futuras a instanciar, limitando el espacio de búsqueda pendiente, y (ii) encontrar las inconsistencias antes de que aparezcan, en el caso de que las instanciações parciales efectuadas se descubran inconsistentes con el resto de variables pendientes. En definitiva, intentan descubrir si la actual asignación localmente consistente de las k variables puede ser extendida a una solución global, provocando en caso contrario un punto de backtracking.

Forward checking (FC). El algoritmo de Look-Forward más común es Forward checking. En este algoritmo, en cada etapa de la búsqueda BT, se comprueba hacia adelante la asignación de la variable actual con todos los valores de las futuras variables que están restringidas con la variable actual. Los valores de las futuras variables que son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios. Si el dominio de una variable futura se queda vacío, la instanciación de la variable actual se deshace y se prueba con un nuevo valor. Si ningún valor es consistente, entonces se lleva a cabo el backtracking cronológico.

De esta manera, el algoritmo FC resulta como si en el algoritmo BT (véase el Algoritmo 10.1), la función *Comprobar* hiciera las siguientes comprobaciones:

1. $\forall i (1 \leq i < k) : (V[k], V[i]) \in c_{ki}$ (* Standard Backtracking *)
2. $\forall f (k < f \leq n), \exists V_f \in D_f : (V[k], V[f]) \in c_{kf}$ (*Forward Chaining*)

Sin embargo, el paso (1) ya no es necesario, ya que debido al proceso FC (paso 2), cada vez que una nueva variable es asignada, sabemos que todos los valores de su dominio son consistentes con los valores de las variables previamente asignadas.

Por ello, no es necesario comprobar cada nueva asignación con las variables previamente asignadas (paso 1).

FC garantiza que, en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura. Además cuando se asigna un valor a una variable, solamente se comprueba hacia adelante con las futuras variables con las que están involucradas. Así, mediante la comprobación hacia adelante, FC puede identificar antes las situaciones sin salida y podar el espacio de búsqueda. El proceso de forward checking puede verse como la aplicación de un simple paso de arco-consistencia entre la variable que acaba de instanciarse y cada una de las variables que quedan por instanciar. El pseudocódigo de algoritmo forward checking es:

1. Seleccionar x_i .
2. Instanciar $x_i \leftarrow a_i : a_i \in D_i$.
3. Razonar hacia adelante (forward-check):
 - Eliminar de los dominios de las variables (x_{i+1}, \dots, x_n) aún no instanciadas, aquellos valores inconsistentes con respecto a la instanciación (x_i, a_i) , de acuerdo al conjunto de restricciones.
4. Si quedan valores posibles en los dominios de todas las variables por instanciar, entonces:
 - Si $i < n$, incrementar i , e ir al paso (1).
 - Si $i = n$, salir con la solución.
5. Si existe una variable por instanciar, sin valores posibles en su dominio, entonces retractar los efectos de la asignación $x_i \leftarrow a_i$. Hacer:
 - Si quedan valores por intentar en D_i , ir al paso (2).
 - Si no quedan valores:
 - Si $i > 1$, decrementar i y volver al paso (2).
 - Si $i = 1$, salir sin solución.

En la Figura 10.11 se puede apreciar la aplicación del proceso FC sobre el ejemplo de CSP correspondiente al problema de las 4-reinas (descrito en la sección 10.3.3). Comprobamos la mejora respecto al algoritmo BT (Figura 10.5), ya que FC requiere generar solo 8 asignaciones. En la primera asignación $x_1 = 1$, se podan (paso 3) los dominios de las variables x_2 , x_3 y x_4 de acuerdo a las restricciones que existen entre dichas variables y la variable x_1 . Tras la asignación $x_2 = 3$, comprobamos (paso 5) que la variable x_4 queda sin valores en el dominio. Esto quiere decir que la instanciación localmente consistente $\{(x_1 = 1), (x_2 = 3)\}$ no puede formar parte de ninguna solución. Ello provoca un punto de backtracking sin necesidad de seguir explorando esta rama. También, comprobamos que cada asignación permite limitar los dominios de las variables pendientes de asignar, acotando de esta forma el espacio

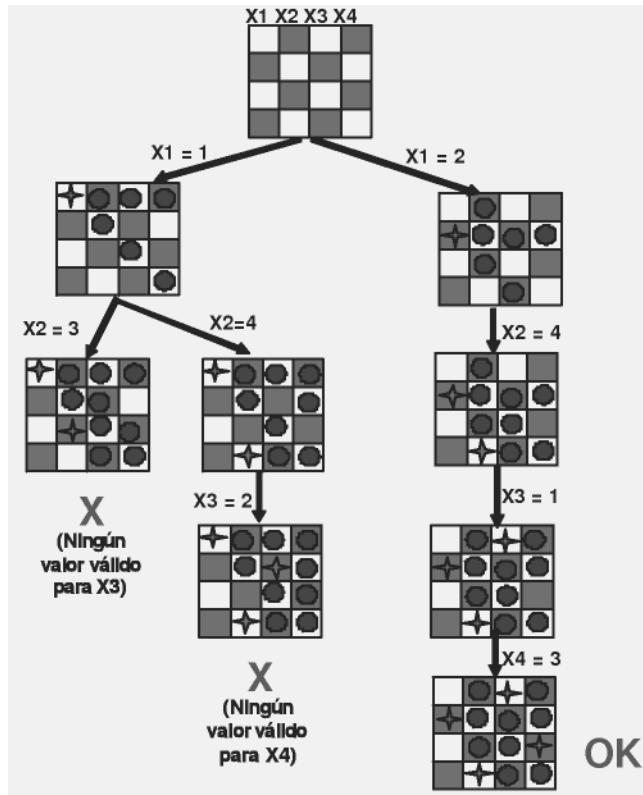


Figura 10.11: Aplicación del Forward Checking al problema de las 4-reinas.

de búsqueda pendiente. Por ejemplo, la asignación $x_1 = 2$ permite limitar los dominios iniciales $\{1, 2, 3, 4\}$ de las variables pendientes a $d_2 = \{4\}$, $d_3 = \{1, 3\}$ y $d_4 = \{1, 3, 4\}$.

Forward checking se ha combinado con algoritmos Look-Backward para generar otros algoritmos *híbridos*. Por ejemplo, *forward checking* con *conflict-directed backjumping* (FC-CBJ) [Prosser, 1993] combina el movimiento hacia adelante de FC con el movimiento hacia atrás de CBJ y de esa manera tiene las ventajas de ambos algoritmos.

Minimal forward checking (MFC) [Dent y Mercer, 1994] es una versión de FC que retrasa llevar a cabo toda la comprobación de la consistencia de FC hasta que es absolutamente necesario. En vez de comprobar hacia adelante la asignación actual contra los valores de las variables futuras, MFC sólo comprueba con los valores de cada variable futura hasta que se encuentra un valor que es consistente. Después, si el algoritmo retrocede llevaría a cabo las comprobaciones con los valores restantes aún no comprobados. Claramente, MFC siempre lleva a cabo a lo sumo el mismo número de comprobaciones que FC. Resultados experimentales han demostrado que la ganancia no supera el 10 %.

Otras versiones de FC están basadas en la idea de desarrollar un mecanismo de poda de dominios que elimina valores no sólo en el nivel actual de búsqueda, sino también en cualquier otro nivel. *Extended forward checking* (EFC) [Bacchus, 2000], tiene la habilidad de podar futuros valores que son inconsistentes con asignaciones hechas antes de la asignación actual pero que no habían sido descubiertas. Otros algoritmos, como el *conflict based forward checking* (CFFC), están basados en la idea de los conflictos, al igual que los algoritmos CBJ y learning analizados en la sección 10.4.3.1, para reforzar a FC. CFFC almacena los conjuntos conflicto y los usa para podar los valores en los niveles pasados de búsqueda. Una diferencia con CBJ es que los conjuntos conflicto se almacenan sobre un valor y no sobre una variable, es decir, cada valor de cada variable tiene su propio conjunto conflicto. Esto permite saltar más lejos que CBJ.

Full Look-Ahead (FLA). Como hemos analizado, el algoritmo FC comprueba únicamente las restricciones entre la variable que acaba de instanciarse con el resto de variables por instanciar. Por ello, mantiene la arco-consistencia sólo entre la variable asignada y el resto de variables. El algoritmo *Full Look-Ahead* (FLA) intenta mantener la arco-consistencia también entre el resto de dichas variables, por lo que también es llamado *maintaining arc consistency* (MAC).

De esta manera, el algoritmo FLA resulta como si en el algoritmo BT (véase el Algoritmo 10.1), la función *Comprobar* hiciera las siguientes comprobaciones (sabiendo que el paso 1 realmente ya no es necesario):

1. $\forall i (1 \leq i < k) : (V[k], V[i]) \in c_{ki}$ (* Standard Backtracking *)
2. $\forall f (k < f \leq n), \exists V_f \in D_f : (V[k], V[f]) \in c_{kf}$ (*Forward Checking*)
3. $\forall p (k < p \leq n), \forall q (k < q \leq n), p \neq q, \exists V_p \in D_p, \exists V_q \in D_q : (V_p, V_q) \in c_{pq}$

El punto (3) comprueba que, tras la poda en los dominios de las variables por instanciar realizadas en FC (en el paso (2)), cada par de variables V_p, V_q que quedan por instanciar, tengan al menos un valor en sus dominios que sean consistentes con la restricción que existe entre ellas.

La ventaja del algoritmo FLA frente al FC es que detecta también los conflictos entre las variables futuras. Por ello, FLA permite podar mucho más los dominios de las variables por instanciar, así como encontrar antes los puntos de backtracking debido a situaciones sin salida. Ello, evidentemente, a costa de un mayor esfuerzo en la propagación de los efectos de cada nueva asignación. Habitualmente, basta con realizar un proceso de FC frente al proceso FLA más costoso. Sin embargo, para problemas altamente restringidos el proceso FLA resulta más conveniente al podar muy fuertemente el espacio de búsqueda.

En la Figura 10.12 vemos la aplicación del proceso FLA sobre el ejemplo de la 4-reinas. Las líneas punteadas corresponden al proceso de propagación. Comprobamos la mejora respecto al algoritmo FC (véase la Figura 10.11), ya que ahora sólo se requiere generar 5 asignaciones.

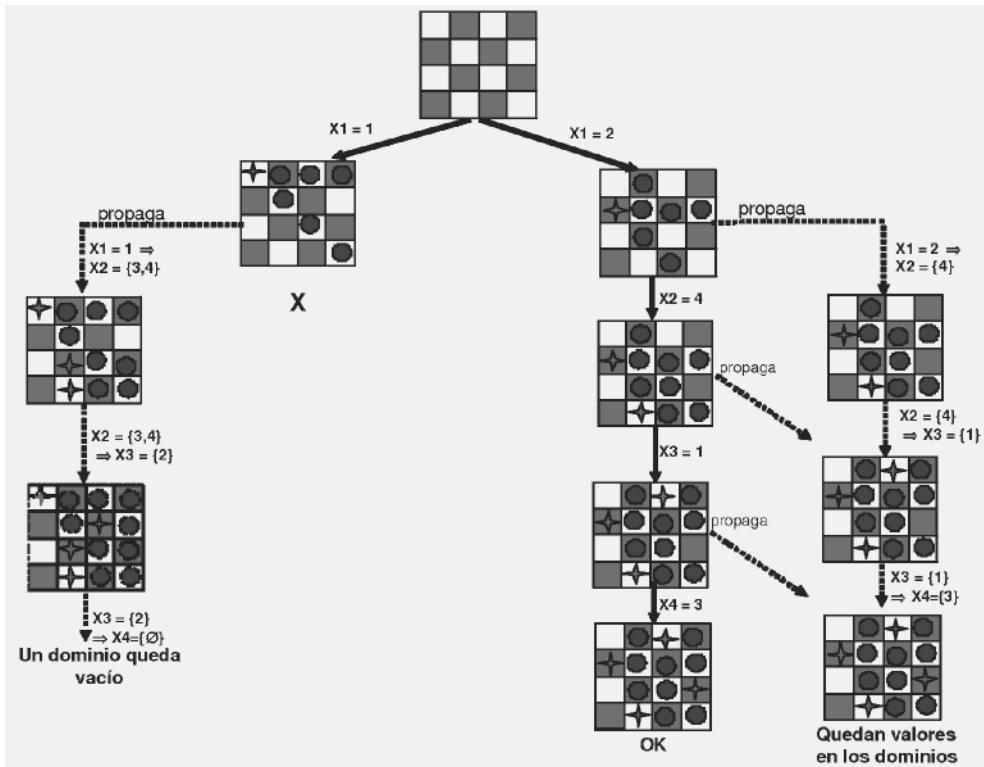


Figura 10.12: Aplicación de Look-Ahead al problema de las 4-reinas.

10.5 Heurísticas de búsqueda

Un algoritmo de búsqueda para la satisfacción de restricciones requiere establecer el orden en el cual se van a estudiar las variables, así como el orden en el que se van a instanciar los valores de los dominios de cada una de las variables. Seleccionar el orden correcto de las variables y de los valores puede mejorar notablemente la eficiencia de resolución. En el método general de búsqueda mediante Backtracking descrito en la sección 10.4.1 (véase el Algoritmo 10.1), la ordenación de variables se corresponde a la ordenación que se realice del vector $V[n]$, mientras que los criterios para la selección de valores se incluirían en la función *Selección* incluida en el Algoritmo 10.1. También puede resultar importante una ordenación adecuada de las restricciones del problema cuando se aplique la función *Comprobar* [Salido y Barber, 2006], aunque se ha realizado mucho menos esfuerzo en el estudio de tales heurísticas. Veamos las más importantes heurísticas de ordenación de variables y de ordenación de valores.

10.5.1 Heurísticas de ordenación de variables

El orden en el cual las variables son asignadas durante la búsqueda puede tener un impacto significativo en el tamaño del espacio de búsqueda explorado. Esto puede ser contrastado tanto empírica como analíticamente. Generalmente las heurísticas de ordenación de variables tratan de seleccionar lo antes posible las variables que más restringen a las demás. La intuición es tratar de asignar cuanto antes las variables más restringidas y de esa manera identificar las situaciones sin salida cuanto antes y así reducir el número de vueltas atrás. La ordenación de variables puede ser estática o dinámica.

10.5.1.1 Ordenación de variables estática

Las heurísticas de *ordenación de variables estática* generan un orden fijo de las variables antes de iniciar la búsqueda, basado en información global derivada de la estructura de la red de restricciones original que representa el CSP. Veamos primero unas ideas iniciales. Una *red de restricciones ordenada* es una red de restricciones cuyos nodos han sido ordenados linealmente. En la Figura 10.13, podemos observar los diferentes órdenes lineales (de arriba a abajo) posibles para una red inicial de tres variables: (a,b,c), (a,c,b), (b,a,c), etc. El *ancho* de una variable en una red de restricciones ordenada es el número de arcos que restringen esa variable con las variables superiores, denominadas *padres*, en el orden lineal de las variables establecido. El ancho de cada red de restricciones ordenada es el máximo ancho de todas sus variables. De esta forma, el *ancho de una red de restricciones* es el mínimo ancho de todas sus posibles ordenaciones lineales. Así, el ancho de la red de restricciones inicial de la Figura 10.13 es 1. Claramente, el ancho de cada red de restricciones depende de su estructura.

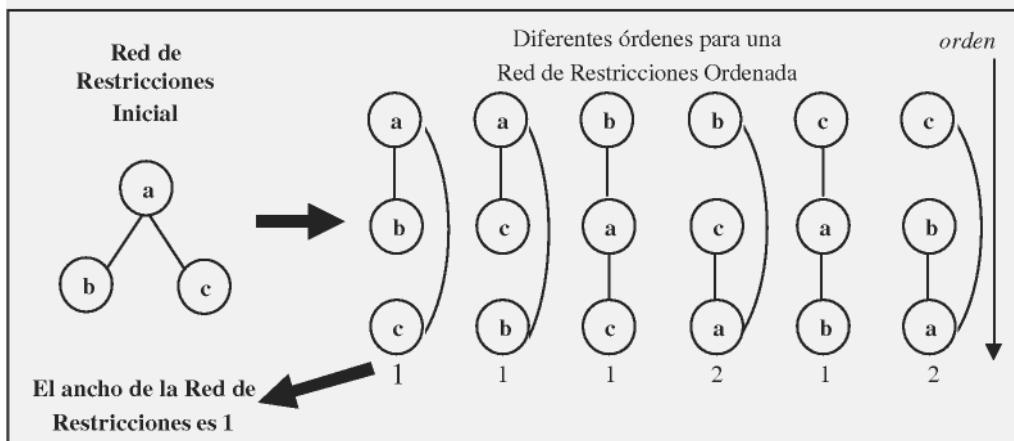


Figura 10.13: Red de restricciones ordenadas.

Si una red de restricciones es fuertemente k -consistente, siendo k mayor que el ancho de la red, entonces existe un orden de asignación de variables que no provocaría vuelta atrás en el proceso de búsqueda. Particularmente, en el caso de redes con estructura de árbol, cuyo ancho es 1, sería posible efectuar una búsqueda sin vuelta atrás si la red fuera arco consistente (2-consistencia). Más concretamente, sería suficiente con mantener la *arco-consistencia direccional* (véase la sección 10.4.2.2), menos costosa que la arco-consistencia. Como ejemplo, en la red de la Figura 10.13, un orden de asignación de variables correspondiente a un ordenamiento de la red de ancho 1 no provocaría vuelta atrás. En un caso general, interesa encontrar el mejor orden para la asignación de variables. Entre las heurísticas más utilizadas podemos mencionar:

- **Minimum Width (MW).** Esta heurística se basa en que la ordenación de variables corresponda a la ordenación lineal de la red que dé lugar a su menor anchura (ancho de la red). Mediante este orden, se persigue que cuando se asigne una variable, sus *padres* estén ya asignados, de forma que las situaciones sin salida puedan identificarse antes y se reduzca el número de vueltas atrás.

Ejemplo 10.5. La red de la Figura 10.14 permite $7!$ (es decir, 5.040) ordenamientos lineales de las variables. El ancho de esta red es 1, que corresponde, entre otras, a la ordenación lineal de las variables indicada en la Figura 10.14. En dicha ordenación, todas las variables tienen un orden 1, por lo que la red inicial tiene un ancho igual a 1. La ordenación de las variables, en el orden lineal de la red, de arriba abajo, resulta (a, b, c, f, g, d, e). Este sería el orden de selección de variables obtenido según el criterio MW. Claramente, al tener la red inicial la estructura de un árbol, su mejor orden corresponde a asignar cuanto antes la variable a, luego la variable b o c, y finalmente, d o e, o bien f o g.

El algoritmo que obtiene el orden de las variables según el método MW selecciona, sobre la red original, la variable con el mínimo número de variables conectadas y la coloca al final del orden. Dicha variable y todos sus arcos adyacentes se eliminan de la red original. Posteriormente, se selecciona la siguiente variable, y el proceso continúa recursivamente.

Ejemplo 10.6. En la Figura 10.15 se muestra la aplicación del método MW a una red de ancho 3. Inicialmente, se selecciona G como la variable menos conectada y se elimina dicha variable y sus relaciones de la red. Posteriormente, se seleccionan {F, E}, después D, C, B y, finalmente A. Podríamos comprobar que este orden da lugar, entre otros, al mínimo ancho de la red:

Orden de Selección:	A →	B →	C →	D →	E →	F →	G
Ancho de la Variable:	0	1	2	3	3	3	3

Luego el orden de selección de variables según MW resultaría (A, B, C, D, E, F, G). Alternativamente, si nos hubieramos decantado por otra ordenación

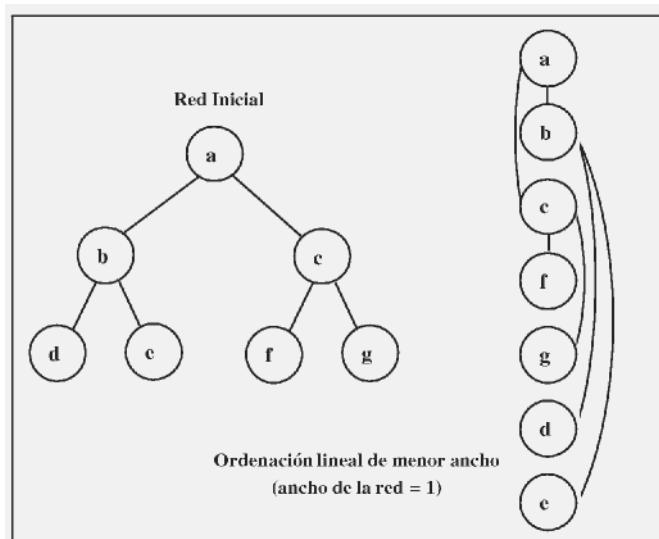


Figura 10.14: Ordenación de una red de mínimo ancho.

(A, B, C, D, E, G, F), no hubiera dado lugar al mínimo orden de la red. El ancho de las variables sería (0,1,2,3,3,2,4) respectivamente, por lo que daría lugar a un ancho 4.

- **Maximum Degree (MD).** Esta heurística ordena las variables en un orden decreciente de su grado en la red de restricciones. El *grado* de una variable se define como el número de variables con las que está conectada. De esta forma, esta heurística selecciona primero las variables de mayor grado, es decir, las más conectadas. En la red de la Figura 10.15, la ordenación de selección sería $(D, A, \{B, C, E, F\}, G)$. Esta heurística intenta seguir el orden de variables de anchura mínima (MW), con un menor coste computacional para el cálculo de dicho orden, aunque no lo garantiza.
- **Mínimum Domain Variable (MDV).** Esta heurística selecciona las variables de acuerdo a la menor cardinalidad de su dominio. Las variables con dominio más pequeño son elegidas antes. Sin embargo, respecto a la aplicación de esta heurística, es preferible su versión dinámica (*minimum remaining values*).

Diversos resultados experimentales sobre heurísticas de ordenación de variables estáticas utilizando CSP generados aleatoriamente concluyen que todas ellas son peores que la heurística *minimum remaining values* (MRV), que es una heurística de ordenación de variables dinámica que veremos a continuación.

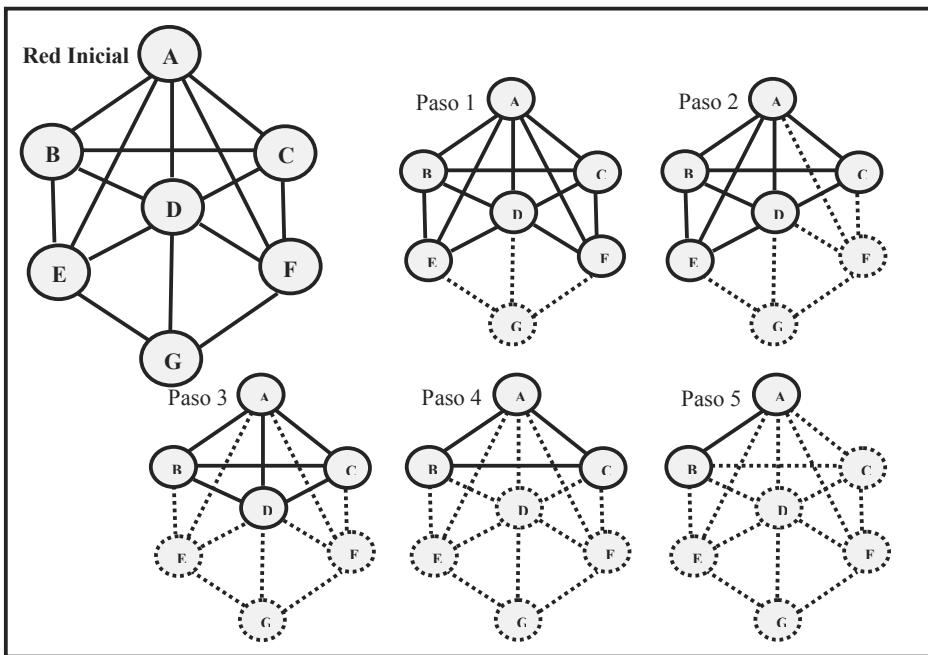


Figura 10.15: Aplicación de la heurística Minimum Width.

10.5.1.2 Ordenación de variables dinámica

El problema de los algoritmos de ordenación de variables estáticos es que no tienen en cuenta los cambios en los dominios o relaciones de las variables causados por la propagación de las restricciones durante la búsqueda. Por ello, estas heurísticas generalmente se utilizan en algoritmos de comprobación hacia atrás (o look-backward) donde no se lleva a cabo la propagación de las instanciaciones que se van realizando.

Las heurísticas de *ordenación de variables dinámica* pueden cambiar el orden de selección de las variables de forma dinámica durante el proceso de búsqueda, cada vez que requiere la instanciación de una variable. La ordenación se basa en las instanciaciones ya realizadas y en el estado de la red en cada momento de la búsqueda.

Se han propuesto varias heurísticas de ordenación de variables dinámicas. Las más comunes se basan en el principio de *primer fallo* que sugiere que “*para tener éxito deberíamos intentar primero donde sea más probable que falle*”. De esta manera, las situaciones sin salida pueden identificarse antes y además se ahorra espacio de búsqueda. De acuerdo a este principio, en cada paso, se selecciona la variable más restringida. Entre las más utilizadas podemos citar:

- **Minimum Remaining Values (MRV).** Esta heurística selecciona, en cada paso, la variable con el dominio de instanciación más pequeño. Esta heurística se basa en la intuición de que si una variable tiene pocos valores de elección

en su dominio, entonces es más difícil encontrar un valor consistente. Cuando se utiliza MRV junto con algoritmos de comprobación hacia atrás (algoritmos look-backward descritos en la sección 10.4.3.1), equivale a una heurística estática que ordena las variables de forma ascendente con la talla del dominio antes de llevar a cabo la búsqueda, y resulta igual por tanto a la heurística MDV. Cuando MRV se utiliza en conjunción con algoritmos look-ahead (véase la sección refsec:alloah), la ordenación se vuelve dinámica. En estos casos, cada nueva instanciación de variable es propagada al resto de la red y puede acotar los dominios de las variables que quedan por instanciar. Así, en cada paso de la búsqueda, la próxima variable a asignar es la variable con el dominio más pequeño.

- **Maximun Cardinality (MC).** Esta heurística selecciona la primera variable arbitrariamente y después en cada paso, selecciona la variable que está relacionada con el mayor número de variables ya instanciadas. La intuición es que resulta la más restringida, al estar relacionada con el mayor número de variables ya instanciadas. Esta heurística resulta similar a la heurística MD previamente definida, pero para el caso dinámico. Como una variación de la misma, podría seleccionarse dinámicamente en cada paso la variable de máximo grado, sin contabilizar las variables ya instanciadas.

10.5.2 Ordenación de valores. Tipos

En comparación con la ordenación de variables, se ha realizado menos trabajo sobre heurísticas para la ordenación de valores. Estas heurísticas tienen como objetivo seleccionar el valor más prometedor para cada variable en su dominio de instanciación. Se aplicarían, por ejemplo, en el paso 2 del algoritmo Forward-Checking descrito en la sección 10.4.3.2. La idea básica es seleccionar el valor de la variable actual que más probabilidad tenga de llevarnos a una solución, es decir, identificar la rama del árbol de búsqueda que sea más probable que obtenga una solución.

La mayoría de las heurísticas propuestas tratan de seleccionar el valor menos restringido de la variable actual, es decir, el valor que menos reduce el número de valores útiles para las futuras variables, o alternativamente, el que deja los dominios mayores. Esto sigue la intuición de que un subproblema es más probable que tenga solución cuantos más valores tengan las variables que quedan por instanciar en sus dominios. Entre las más utilizadas podemos mencionar:

- **min-conflicts.** Es una de las heurísticas de ordenación de valores más conocida. Esta heurística ordena los valores de acuerdo a los conflictos que generan con las variables aún no instanciadas. Esta heurística asocia a cada valor a de la variable actual, el número total de valores en los dominios de las futuras variables adyacentes con la actual que son incompatibles con a . El valor seleccionado es el asociado a la suma más baja.
- **max-domain-size.** Alternativamente a la anterior, esta heurística selecciona el valor de la variable actual que deja el máximo dominio en las variables futuras.

- **weighted-max-domain-size.** Esta heurística especifica una manera de romper empates en el método anterior, en el caso de que existan varios valores de la variable actual que dejen el mismo máximo dominio en las variables futuras. Entonces, se basa en el número de futuras variables que tienen la mayor talla de dominio. Por ejemplo, si un valor a_i deja cinco variables futuras con dominios de seis elementos (y el resto con dominios mayores de seis), y otro valor a_j deja siete variables futuras también con dominios de seis elementos, en este caso se selecciona el valor a_i .
- **point-domain-size.** Esta heurística asigna un peso (unidades) a cada valor de la variable actual dependiendo del número de variables futuras que se quedan con ciertas tallas de dominios. Por ejemplo, para cada variable futura que se queda con un dominio de talla 1 debido al valor a_i , se añaden 8 unidades al peso de a_i , para cada variable futura que se queda con un dominio de talla 2 se añaden 7 unidades, etc. De esta manera se selecciona el valor con el menor peso.
- **Supervivencia.** Esta heurística propone una variación de la idea anterior de la heurística *min-conflicts*. De acuerdo a dicha heurística, cuando se cuenta el número de valores incompatibles en el dominio de una futura variable x_k , debido a la elección del valor a_i para x_i , éste número se divide por la talla del dominio de x_k . Esto da el porcentaje de los valores útiles que pierde x_k debido al valor a_i que actualmente estamos examinando. De nuevo, los porcentajes se añaden para todas las variables futuras con las que está relacionada x_i y se selecciona el valor más bajo que se obtiene en todas las sumas.
- **max-promise.** En esta heurística, para cada valor a_i de la variable x_i se cuenta el número de valores que hay compatibles con a_i en cada variable adyacente futura, y se toma el producto de las cantidades contadas. Este producto se llama *promesa del valor*. De esta manera, se selecciona el valor con la máxima promesa. Al usar el producto en vez de la suma del número de valores compatibles, la heurística *max-promise* trata de seleccionar el valor que deja un mayor número de soluciones posibles después de que este valor se haya asignado a la variable actual. De esta manera, la promesa de cada valor representa una cota superior del número de soluciones diferentes que pueden existir después de que el valor se asigne a la variable.

10.6 Extensiones de CSP

En esta sección se presentan algunas de las extensiones más relevantes a la metodología CSP, orientadas a tipologías de problemas más concretas o que plantean requerimientos adicionales.

10.6.1 CSP no binarios

Gran parte de las técnicas de satisfacción de restricciones se centran en problemas binarios, en el caso de CSP discretos, y en problemas ternarios, en el caso de CSP continuos. La razón básica de tratar con restricciones binarias y ternarias es por la simplicidad que supone comparado con las restricciones n -arias, y también por el hecho de que todo problema de satisfacción de restricciones n -arias puede ser transformado a un problema binario o a uno ternario equivalente, bajo la definición de *equivalencia de red*.

Principalmente, hay dos técnicas para convertir las restricciones no binarias a binarias en CSP discretos:

- **Codificación dual**, introducida por Dechter y Pearl, en la cual las restricciones del problema original se transforman en variables del nuevo problema, y viceversa. Las variables que se generan de las restricciones originales se denominan *variables duales*, y las variables del problema original se denominan *variables originales*. El dominio de cada variable dual es el conjunto de tuplas que satisfacen la restricción original que la genera. Además, hay una restricción binaria entre dos variables duales v_c y v'_c si y solamente si las restricciones originales c y c' comparten una o más variables. Las nuevas restricciones binarias se denominan restricciones duales y prohíben pares de tuplas donde las variables compartidas tomen valores diferentes.

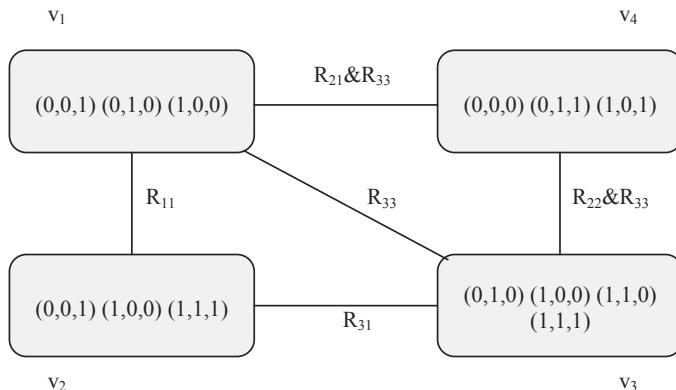


Figura 10.16: Ejemplo de codificación dual de un CSP no binario.

Si consideremos el siguiente ejemplo, tomado de [Stergiou y Walsh, 1999], con seis variables con dominios $\{0, 1\}$ y cuatro restricciones: $x_1 + x_2 + x_6 = 1$, $x_1 - x_3 + x_4 = 1$, $x_4 + x_5 - x_6 \geq 1$ y $x_2 + x_5 - x_6 = 0$. La codificación dual representa este problema con cuatro variables duales, una para cada restricción. Los dominios de estas variables duales son las tuplas que satisfacen las respectivas restricciones. Por ejemplo, la variable dual v_3 asociada a la tercera restricción, $x_4 + x_5 - x_6 \geq 1$, tiene como dominio $(0, 1, 0), (1, 0, 0), (1, 1, 0), (1, 1, 1)$, ya que estas son las tuplas de valores para (x_4, x_5, x_6) que satisfacen la restricción. La codificación dual del

problema se muestra en la Figura 10.16. R_{ij} es la restricción binaria sobre un par de tuplas que se satisface si y sólo si el i -ésimo elemento de la primera tupla es igual al j -ésimo elemento de la segunda tupla. Entre v_3 y v_4 hay una restricción de compatibilidad para asegurar que las dos variables originales en común, x_5 y x_6 tienen los mismos valores. Esta restricción permite solamente aquellos pares de tuplas que concuerden con los elementos segundos y terceros, es decir $(1, 0, 0)$ para v_3 y $(0, 0, 0)$ para v_4 .

- **Codificación de variables ocultas**, que procede de Peirce, quien, en 1933, probó formalmente en el campo de la filosofía lógica que las relaciones binarias y no binarias tienen el mismo poder de expresividad. Su método para representar una relación no binaria mediante una colección de restricciones binarias formó las bases del método de las variables ocultas. En la codificación de variables ocultas, el conjunto de variables está formado por todas las variables del CSP no binario original más un nuevo conjunto de variables duales que llamaremos *variables ocultas*. Al igual que la codificación dual, cada variable oculta corresponde a una restricción en el problema original. De nuevo, el dominio de cada variable oculta consta de las tuplas que satisfacen la restricción original. Para cada variable oculta v_c , hay una restricción binaria entre la variable v_c y cada una de las variables originales x_i que están involucradas en la correspondiente restricción original c . Cada restricción especifica que la tupla asignada a v_c es consistente con el valor asignado a x_i .

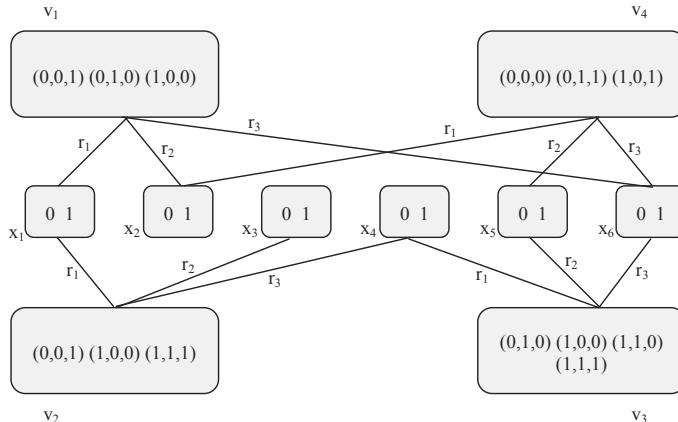


Figura 10.17: Ejemplo de codificación de variables ocultas de un CSP no binario.

Considerando de nuevo el ejemplo anterior presentado en [Stergiou y Walsh, 1999], donde el problema consta de seis variables y cuatro restricciones no binarias, en la codificación de variables ocultas hay, además de las seis variables originales con dominios $\{0, 1\}$, cuatro variables duales con los mismos dominios que en la codificación dual. Por ejemplo, la variable dual asociada con la tercera restricción $x_4 + x_5 - x_6 \geq 1$, tiene como dominio $(0, 1, 0), (1, 0, 0), (1, 1, 0), (1, 1, 1)$.

Además, ahora hay restricciones de compatibilidad entre v_3 y x_4 , entre v_3 y x_5 y entre v_3 y x_6 , ya que estas son las variables que están involucradas en la tercera restricción. La codificación de variables ocultas de este problema se muestra en la Figura 10.17. La restricción binaria r_i actúa sobre una tupla y un valor, dando verdadero si y solo si el elemento i -ésimo de la tupla es igual al valor. Por ejemplo, la restricción de compatibilidad r_1 entre v_3 y x_4 es cierta si y sólo si el primer elemento de la tupla asignada a v_3 es igual al valor de x_4 .

En el caso de CSP numéricos o continuos, se pueden transformar las restricciones n -arias en restricciones ternarias, según el método descrito por Lottaz, de forma que se mejora la eficiencia de los algoritmos de consistencia. Esta transformación sólo es posible si se permite la introducción de variables auxiliares.

La transformación de CSP n -arios en CSP binarios equivalentes permite la aplicación de las diversas técnicas conocidas para CSP binarios. Sin embargo, se pierde la legibilidad de las restricciones, originalmente adquiridas desde el análisis del problema como restricciones n -arias. Por otra parte, resultan más difíciles de aplicar heurísticas de búsqueda orientadas al dominio, al haber perdido las variables y las restricciones su equivalencia con el problema original. Por ello, se mantiene el interés de trabajar con CSP n -arios.

10.6.2 CSP distribuidos (DisCSP)

En el ámbito de los sistemas multiagente, los problemas de satisfacción de restricciones se pueden manejar mediante el uso de agentes inteligentes que se encargan de controlar subconjuntos de variables y restricciones del problema. Si todo el conocimiento sobre el problema puede ser concentrado en un único agente, este agente podrá resolver el problema mediante el uso de técnicas centralizadas de satisfacción de restricciones. Sin embargo, obtener una solución de forma centralizada es con frecuencia inadecuado o incluso imposible. Algunas razones por las que son deseables el uso de métodos distribuidos son [Faltings y Yokoo, 2005]:

- **Coste de crear una autoridad central.** Un CSP puede ser distribuido de forma natural en un conjunto de agentes concretos. En tales casos, el uso de una autoridad central para resolver el problema requiere establecer un elemento adicional que no estaba presente en la arquitectura del problema.
- **Privacidad/seguridad.** Las restricciones que maneja cada agente puede ser información estratégica que no debería ser revelada a la competencia o incluso a una autoridad central. La privacidad es más fácil de mantener en resoldedores distribuidos.
- **Robustez frente a fallos.** El fallo de un servidor central puede ser fatal. En un método distribuido, el fallo de un agente puede ser menos crítico y otros agentes podrían ser capaces de encontrar una solución al problema.

Adicionalmente, pueden existir características de jerarquía, cooperación, etc., en un problema distribuido que se puede representar de una manera más eficiente mediante una arquitectura de CSP distribuida. De esta manera, surge la necesidad de

distribuir el problema en una serie de subproblemas que cumplan con los anteriores criterios y que, además, sean más fáciles de resolver. Sin embargo, una de la complejidad fundamental en la resolución de los DisCSP es el sobrecoste que supone el intercambio de mensajes entre los agentes. Por otra parte en los problemas distribuidos no se pueden aplicar heurísticas e inferencias de CSPs centralizados.

Un CSP Distribuido (DisCSP) es un CSP donde las variables y las restricciones son distribuidas entre múltiples agentes. Cada agente en un DisCSP tiene algunas variables y trata de determinar sus valores. Sin embargo hay restricciones inter-agentes y la asignación de valores debe satisfacer estas restricciones. La búsqueda que se lleva a cabo para resolver un DisCSP puede ser llevada a cabo por un agente central, aunque en la mayoría de los casos la búsqueda se lleva a cabo de forma distribuida a través del intercambio de mensajes entre los diferentes agentes. Existen dos aproximaciones básicas para la búsqueda de soluciones en DisCSP. La primera de ellas es la llamada *Synchronous Backtracking* (SBT). Para ello, se ordenan inicialmente todos los agentes (A_1, A_2, \dots, A_n). El primer agente A_1 obtiene una asignación consistente para sus variables y envía al siguiente agente dicha asignación, y así sucesivamente. De esta forma, cada agente espera un mensaje del agente anterior para empezar la búsqueda. Cuando un agente A_i recibe el mensaje, intenta obtener una asignación a sus variables de tal manera que se cumplan todas las restricciones, incluyendo las variables asignadas en los agentes previos. Si un agente A_i no pudiera encontrar una asignación consistente, envía un mensaje al agente anterior A_{i-1} , a fin de pedirle una asignación alternativa. La solución global se alcanza cuando el último agente A_n encuentra una asignación para sus variables. En cambio, no existe solución si el primer agente A_1 recibe un mensaje de asignación alternativa de A_2 , y no puede obtenerla.

En un esquema SBT, en principio, los agentes no se ejecutan concurrentemente. Sin embargo, se han planteado esquemas que superan este inconveniente, de forma que cada agente A_i continúa buscando soluciones alternativas a la previamente encontrada, a fin de poder ofrecer dicha instanciación rápidamente al agente siguiente A_{i+1} , si recibe un mensaje para ello. Junto a este esquema, también se pueden plantear las típicas alternativas de un proceso de Backtracking como el conocido *BackJumping*.

La segunda alternativa fue llevada a cabo por Yokoo y otros [Yokoo y otros, 1998] en sus algoritmos de *backtracking asíncrono*, donde el DisCSP se resuelve mediante el intercambio de mensajes de forma asíncrona. Ello permite a los agentes actuar concurrentemente sin un control global, garantizando la completitud del proceso. Para ello, se asume una ordenación de prioridades entre los agentes y cada agente es responsable de forzar todas las restricciones entre sus variables y las variables que sean propiedad de agentes de rango mayor. A partir de este planteamiento, y utilizando la ordenación de prioridades, se desarrollan dos algoritmos de búsqueda distribuida, *asynchronous backtracking* (ABT) y *asynchronous weak-commitment* (AWC). La principal diferencia entre ellos es el manejo de las situaciones sin salida (*dead-end*), es decir, cuando un agente no puede encontrar una asignación consistente para sus variables. En ABT, el agente retrocede y envía un mensaje a los agentes de prioridad superior alegando que la combinación de asignaciones de valores a las variables de rango superior no puede formar parte de una solución en sus variables, y solicitando que se cambie una asignación. Por el contrario, en AWC, un agente utiliza una técnica llamada *com-*

promiso débil, donde un agente renuncia a la tentativa de satisfacer sus restricciones y las delega a otros agentes levantando su propia prioridad. Mientras hace esto, un agente puede enviar instancias no consistentes a otros agentes por lo que ellos evitarán asignaciones de valores especificadas en las instancias no consistentes. Resultados experimentales muestran que AWC es significativamente mejor que ABT en la obtención de soluciones en algunas instancias de DisCSP difíciles. Otras aproximaciones para el manejo de DisCSP distintas se pueden ver en [Faltings y Yokoo, 2005].

10.6.3 CSP temporales y CSP dinámicos

Un CSP temporal es un caso particular de CSP donde las variables representan una primitiva temporal (puntos de tiempo, intervalos, o duraciones temporales), estableciéndose restricciones temporales entre sí. Constituyen una tipología de CSP de gran aplicación (bases de datos, lenguaje natural, scheduling, etc.). Los modelos básicos de CSP temporales son binarios, y se basan en álgebras temporales. Un álgebra temporal define, fundamentalmente: (i) el conjunto de restricciones sobre las variables ($x_i c_{ij} x_j$), y (ii) las operaciones básicas entre las mismas. Estas operaciones son, principalmente:

- **Adición** de una nueva restricción sobre un mismo par de variables:

$$(x_i \{c_{ij}\} x_j) \oplus (x_i \{c'_{ij}\} x_j) = x_i \{c_{ij} \oplus c'_{ij}\} x_j$$

- **Combinación** (transitiva), que permite obtener la relación c_{ik} a partir de c_{ij} y c_{jk} :

$$(x_i \{c_{ij}\} x_j) \otimes (x_j \{c_{jk}\} x_k) = x_i \{c_{ij} \otimes c_{jk}\} x_k$$

- Otras operaciones son la inclusión: ($c_{ij} \subset c'_{ij}$), la negación ($\neg c_{ij}$), etc.

Los principales CSP temporales son:

- **Álgebra de Puntos** [Vilain y Kautz, 1986], que establece restricciones temporales cualitativas y disyuntivas entre puntos de tiempo: $t_i \{<, =, >\} t_j$. En la Figura 10.18 se muestran las tablas para las operaciones \oplus y \otimes de este álgebra.
- **Álgebra de Intervalos** [Allen, 1983], que establece restricciones temporales cualitativas y disyuntivas entre intervalos: $I_i \{before, meets, overlaps, during, starts, finishes, equal\} I_j$, así como sus inversas.
- **Álgebra Métrica entre puntos** [Dechter y otros, 1991], que establece restricciones temporales métricas y disyuntivas entre puntos de tiempo: $t_j - t_i \in \{[a_1, b_1], [a_2, b_2], \dots, [a_n, b_n]\}$, donde a_i, b_i representan puntos extremos de intervalos métricos.

Estas álgebras permiten especificar las típicas restricciones de scheduling respecto a la compartición no simultánea de recursos. Por ejemplo, si las tareas W_1 y W_2 deben compartir, de forma no simultánea, un recurso común, debemos especificar que su ejecución no debe solaparse en el tiempo. Es decir:

1. Mediante restricciones sobre los intervalos de ocurrencia de las tareas:

$$I_{w_1} \{before, after\} I_{w_2}$$

2. Mediante restricciones sobre los puntos de tiempo extremos:

$$End_{w_1} < Start_{w_2} \vee End_{w_2} < Start_{w_1}$$

3. O bien, conociendo la duración de las tareas, podemos utilizar un modelo métrico:

$$Start_{w_1} - Start_{w_2} \in \{-\infty, -dur_{w_1}\}, [dur_{w_2}, \infty\}$$

\oplus	<	\leq	>	\geq	=	\neq	?
<	<	<	\emptyset	\emptyset	\emptyset	<	<
\leq	<	\leq	\emptyset	=	=	<	\leq
>	\emptyset	\emptyset	>	>	\emptyset	>	>
\geq	\emptyset	=	>	\geq	=	>	\geq
=	\emptyset	=	-	=	=	-	=
\neq	<	<	>	>	\emptyset	\neq	\neq
?	<	\leq	>	\geq	=	\neq	?

\otimes	<	\leq	>	\geq	=	\neq	?
<	<	<	?	?	<	?	?
\leq	<	\leq	?	?	\leq	?	?
>	?	?	>	>	>	?	?
\geq	?	?	>	\geq	\geq	?	?
=	<	\leq	>	\geq	=	\neq	?
\neq	?	?	?	?	\neq	?	?
?	?	?	?	?	?	?	?

Figura 10.18: Tablas para las operaciones \oplus y \otimes en el álgebra de puntos (\emptyset es la inconsistencia y ? representa $\{\leq\}$).

Otras álgebras temporales establecen restricciones entre duraciones, entre puntos e intervalos, y entre puntos, duraciones e intervalos, etc. Estos modelos plantean ya restricciones ternarias.

Los CSP temporales binarios se pueden representar mediante una red de restricciones temporales (Temporal Constraint Network o TCN), como un grafo dirigido, donde una restriccion entre x_i, x_j , implica una restriccion simétrica entre x_j, x_i .

Consideremos el siguiente ejemplo [Dechter y otros, 1991]: “Juan va de su casa al trabajo en coche (30-40 minutos) o en tren (al menos una hora). Luis va en coche (20-30 minutos) o en metro (40-50 minutos). Hoy Juan parte de casa entre las 8:10 y las 8:20 y Luis llega al trabajo entre las 9:00 y las 9:10. Además, sabemos que Juan llegó al trabajo entre 10 y 20 minutos después de que Luis saliera de casa”. Esta información se puede modelar mediante un CSP métrico entre puntos, representado en la red temporal de la Figura 10.19:

- **Variables:** $T_0, T_1, T_2, T_3, T_4 \in \{0, \dots, 3600\}$

- **Restricciones:**

- $10 \leq T_1 - T_0 \leq 20$
- $30 \leq T_2 - T_1 \leq 40 \vee 60 \leq T_2 - T_1$
- $10 \leq T_2 - T_3 \leq 20$
- $20 \leq T_4 - T_3 \leq 30 \vee 40 \leq T_4 - T_3 \leq 50$

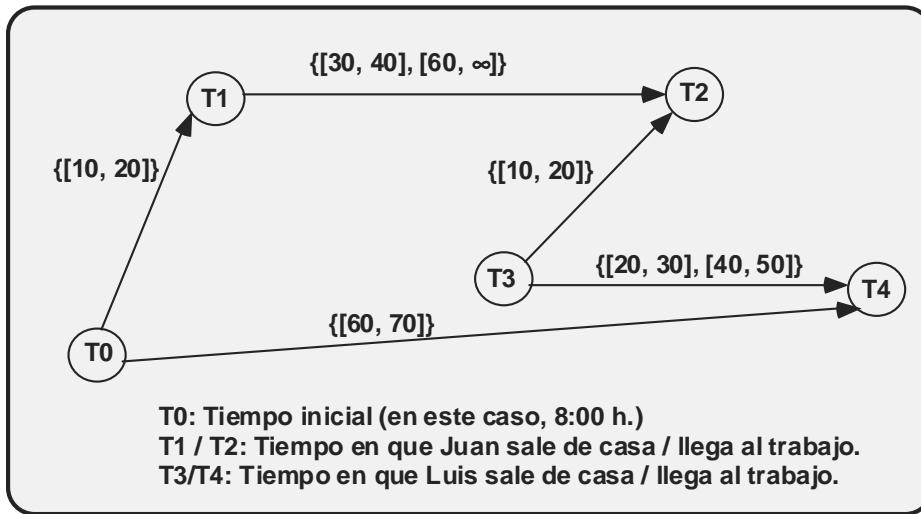


Figura 10.19: Una red temporal.

Las restricciones representadas en la Figura 10.19 corresponden a la información explícitamente conocida del problema. Ante ella, se pueden hacer las siguientes preguntas: ¿Esta información es consistente?, es decir, ¿tiene solución? ¿Es posible que Juan haya usado el tren y Luis haya usado el Metro? ¿Cuáles son los posibles tiempos en los que Luis pudo haber salido de casa? ¿Quién llega antes al trabajo, Juan o Luis?, etcétera.

Estas preguntas se pueden responder mediante los procesos inferenciales descritos en la sección 10.4.2. Particularmente, resulta útil la adaptación del algoritmo de senda consistencia descrito en Algoritmo 10.3, de forma que la clausura de las restricciones en cada subred de 3 nodos (función *Revisar*) se realice mediante la operación de combinación (\otimes). Ello da lugar al algoritmo de Clausura Transitiva (*Transitive Closure Algorithm*, TCA) cuya función *Revisar* se muestra en el Algoritmo 10.4.

Algoritmo 10.4 Función Revisar del algoritmo de clausura transitiva.

```

función Revisar( $i, j, k$ )
  inicio
    cambiado := false;
     $c'_{ij} \leftarrow c_{ij} \oplus (c_{ik} \otimes c_{kj})$ 
    si  $c_{ij} = \emptyset$  entonces
      Inconsistente;
    fin si
    si  $c'_{ij} \subset c_{ij}$  entonces
       $c_{ij} := c'_{ij}$ ; ; Acota la restricción  $c_{ij}$ 
      cambiado := true
    fin si
    devolver cambiado;
  fin Revisar;
```

Como se puede apreciar, el algoritmo TCA obtiene una red senda-consistente, donde:

$$\forall c_{ij}, c_{ik}, c_{kj} \subseteq CSP : c_{ij} \leftarrow c_{ij} \oplus (c_{ik} \otimes c_{kj})$$

La aplicación de este proceso al CSP de la Figura 10.19, obtendría una red clausurada en la que se han hecho explícitas las restricciones implícitamente contenidas en el problema (así fue el caso de la red de la Figura 10.10). La red mínima equivalente se muestra en la Figura 10.20, donde las nuevas restricciones derivadas permitirían responder a algunas de las preguntas anteriores.

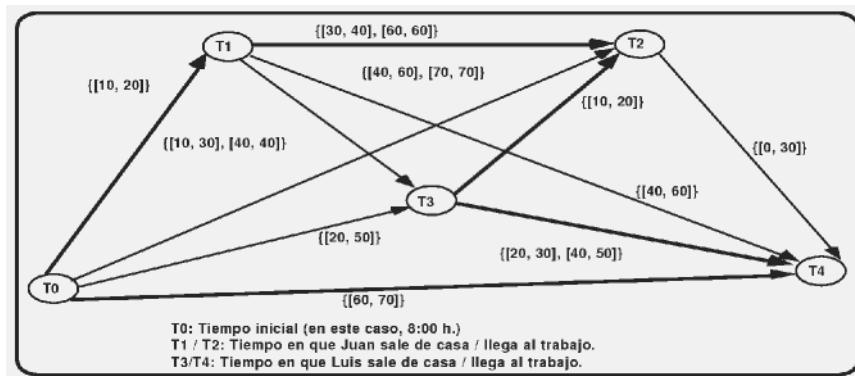


Figura 10.20: Red Mínima correspondiente a la red de la Figura 10.19.

Especialmente en el marco del dominio temporal, y más concretamente en su aplicación a problemas que evolucionan (planificación, scheduling, etc.), aparecen los *CSP Dinámicos (DCSP)*. Un DCSP se define como una secuencia de CSP, donde cada uno de ellos corresponde a un cambio en el CSP previo [Dechter y Dechter, 1988]. Así, en un DCSP pueden añadirse o retractarse restricciones de forma dinámica, lo cual puede ocurrir durante el proceso de comprobación de la consistencia, durante la búsqueda de soluciones, o bien una vez obtenida una solución. Todos los CSP estáticos en la secuencia de un DCSP deben ser resueltos para solucionar el DCSP. El problema particular en los DCSP reside en encontrar sucesivamente, si existen, nuevas soluciones después de cada modificación. Esto se denominada *problema de mantenimiento de la solución*, que emplea técnicas específicas (obtención de soluciones próximas, reparación de conflictos, etc.) a fin de evitar sucesivos, redundantes e inefficientes, procesos de búsqueda.

10.6.4 Otras extensiones

Para la resolución de un CSP pueden aplicarse también *técnicas estocásticas*, técnicas de búsqueda no sistemáticas e incompletas, incluyendo heurísticas. Ejemplos de este tipo de técnicas son hill-climbing, búsqueda tabú, enfriamiento simulado, algoritmos genéticos, etc. Estas técnicas pueden considerarse como adaptativas, en el sentido

de que ellas comienzan su búsqueda en un punto aleatorio del espacio de búsqueda y lo modifican repetidamente, utilizando heurísticas, hasta que se alcanza una solución tras un cierto número de iteraciones. Estos métodos son, generalmente, robustos y buenos para encontrar soluciones optimizadas en espacios de búsqueda grandes, y han probado su utilidad en la resolución de CSP complejos. La característica estocástica de estos métodos introduce un aspecto aleatorio y pueden ocurrir situaciones donde el proceso de búsqueda se encuentre en porciones erróneas del espacio de búsqueda. Esto generalmente requiere que el sistema reinicie su ejecución empezando en otro punto de partida aleatorio.

Otra extensión de los CSP lo constituyen los *CSP difusos (FCSP)* [Dubois y otros, 1993] que resultan de utilidad en dominios en los que no hay una descripción suficientemente concreta de las restricciones (por ejemplo: “La acción duró *casi* una hora”). Haciendo uso de los conceptos de la lógica difusa, en los FCSP se introduce el concepto de restricción difusa, como una restricción a la que se asocia un función de pertenencia ρ_R . El concepto de satisfacibilidad resulta ahora una cuestión de grado. Una asignación a las variables $((x_1, a_1), \dots, (x_i, a_i))$ es una ρ -solución ($\rho \in [0, 1]$) si satisface la combinación de todas las restricciones con grado ρ . De esta forma, puede relajarse el grado de cumplimiento de restricciones que impiden la obtención de soluciones.

Además del concepto de restricciones difusas, también podría hablarse de restricciones fuertes y restricciones débiles (*hard and soft constraints*). Mientras que las restricciones fuertes deben cumplirse, cabe incluir preferencias o alguna relajación en el cumplimiento de las soft-constraints. Una forma de aproximación a este tipo es el uso de CSP valuados (*Weigthed-CSP*) [Schiex y otros, 1995]. En ellos, las relaciones entre variables incluyen funciones de coste que expresan el grado de satisfacción de las restricciones que vinculan dichas variables.

Otra extensión compleja de los CSP son los denominados *CSP numéricos*. Generalmente n -arios, los CSP numéricos están ligados a una modelización matemática de los problemas (restricciones numéricas, polinomiales, etc.), incluyen dominios infinitos, y requieren la aplicación de técnicas matemáticas específicas para su resolución. Sin embargo, los algoritmos tipo *simplex* son aplicables para restricciones lineales, pero en cambio no resultan tan adecuados para restricciones no lineales o con dominios discretos.

Existe una clara relación de la metodología CSP con la *programación por restricciones* donde las relaciones entre las variables también pueden establecerse en forma de restricciones embebidas en un lenguaje de programación y, particularmente, con la programación lógica. De esta forma, la *programación lógica con restricciones* (Constraint Logic Programming, CLP) puede verse como un CSP donde las restricciones se reescriben como predicados y la unificación en LP se sustituye por la resolución por coerción en un dominio específico (satisfacción de restricciones) [Hentenryck, 1989]. En relación con ello, en el *problema de satisfacibilidad* (SAT), las variables se instancian en el dominios de los valores booleanos $\{True, False\}$ y las relaciones utilizan los operadores lógicos *and*, *or* y *not*. En su caso más simple, el problema es encontrar una asignación a las variables que haga cierta una expresión.

10.7 Lecturas recomendadas

Un texto completo sobre la metodología CSP se encuentra en [Dechter, 2003]. También son interesantes las referencias [Apt, 2003; Frühwirth y Abdennadher, 2003; Marriott y Stuckey, 1998]. De forma más resumida, aunque cubriendo una amplia tipología de CSP y sus aplicaciones, cabe consultar la monografía contenida en [Barber, 2003]. Por otra parte, bastante completa y frecuentemente renovada resulta “On-Line Guide To Constraint Programming” de R. Barták, accesible en [<http://kti.mff.cuni.cz/bartak/constraints/index.html>]. En relación con ello, una visión general de los diferentes tipos de CSP y sus procesos de consistencia puede verse en [Barták, 2001]. Puede encontrarse una revisión de las aplicaciones CSP en [Wallace, 1995]. Finalmente, cabe destacar el interés de diversas páginas web de grupos de trabajo en el área, con aplicaciones, herramientas, benchmarks, bibliografía, tutoriales, etc.

10.8 Resumen

En este capítulo se ha hecho una revisión de las principales técnicas que subyacen en la metodología CSP, para la resolución de un amplio tipo de problemas combinatorios. Las etapas fundamentales en la resolución de problemas CSP son (Figura 10.21):

1. La modelización del problema, mediante variables, dominios y restricciones, donde es importante la capacidad expresiva de las restricciones a fin de que puedan captar toda la información relevante del problema.
2. La aplicación de métodos inferenciales, que deducen nueva información (nuevas restricciones implícitas). Ello permite poder contestar a preguntas sobre el problema, así como acotar el espacio de búsqueda, por lo que algún proceso de k -consistencia suele realizarse antes del proceso de búsqueda. En las técnicas inferenciales, es importante su capacidad deductiva *versus* coste computacional. Dependiendo del proceso se obtienen diversos niveles de consistencia, así como mayor información deducida.
3. La aplicación de técnicas de búsqueda de soluciones al problema. Con un coste, en general, *NP-completo* (*NP-duro* en el caso de optimización) estas técnicas se integran con las técnicas inferenciales (técnicas híbridas), y pueden incluir heurísticas, de ordenación de variables o de valores, que permitan la obtención de soluciones en un tiempo razonable.

Tras la revisión de los principales conceptos y algoritmos de cada una de estas etapas, se ha presentado una breve referencia a algunas de las extensiones o tipologías concretas de CSP. El conocimiento de las técnicas CSP permite disponer de un conjunto de técnicas que permiten abordar la resolución de un amplio tipo de problemas.

La aplicación de los CSP es muy amplia, especialmente en la ingeniería, medicina e informática. Esta metodología resulta útil para solucionar problemas combinatorios,

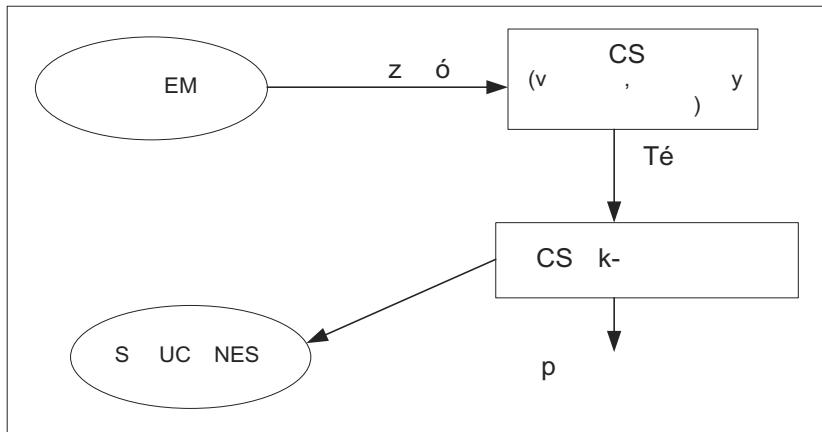


Figura 10.21: Etapas para la resolución de problemas mediante CSP.

temporales, de configuración etc., cuya resolución esté sujeta a la satisfacción de un conjunto de restricciones entre las variables del problema. Han demostrado su utilidad en aplicaciones de investigación operativa (generación de horarios, scheduling, etc.), bioinformática (identificación y ordenación de secuencias genéticas), ingeniería electrónica (diseño de circuitos), telecomunicaciones (asignación de frecuencias), informática (bases de datos y sistemas de recuperación de la información, lenguaje natural, planificación, etc.), en problemas de diseño y configuración, empaquetamiento, etc. En [Wallace, 1995] pueden verse algunas de las técnicas asociadas a estas aplicaciones.

La resolución de problemas mediante un CSP puede considerarse como un método fundamentalmente declarativo. De esta forma, basta especificar el problema a resolver como un CSP. Para resolver un CSP y obtener soluciones pueden utilizarse alguna de las múltiples herramientas disponibles, tanto comerciales, software libre, o librerías con procedimientos especializados en el manejo de restricciones. Estas herramientas existen en diferentes lenguajes, particularmente C++, Java o Lisp. Las herramientas CSP permiten la edición de la especificación del problema (definición de las variables, sus dominios y restricciones) e incluyen potentes métodos de inferencia y búsqueda que nos devuelven una, varias, o todas las soluciones al problema.

10.9 Ejercicios resueltos

10.1. Dada la red inicial de la Figura 10.22, donde en cada nodo se representa el dominio de la variable, obténgase la red arco consistente equivalente.

Solución: Para obtener una red arco-consistente, debemos forzar la arco-consistencia de cada arco de la red, $C = \{c_{12}, c_{23}\}$, mediante el algoritmo descrito en la sección 10.4.2.2, teniendo en cuenta que cada arco implica un arco simétrico.

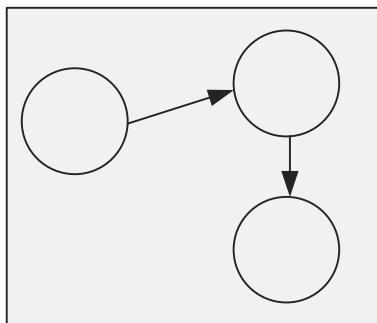


Figura 10.22: Red inicial del Ejercicio 1.

De esta forma, al hacer c_{12} arco-consistente, se limita $d_1 = \{6, 10\}$ y $d_2 = \{5, 9\}$. Se añadiría c_{23} a C , pero ya está.

Para hacer c_{23} arco-consistente, se limita $d_2 = \{9\}$ y $d_3 = \{8\}$. Se añade c_{12} a C . Para hacer c_{12} arco-consistente, $d_1 = \{10\}$.

Puede verse que los dominios de la red quedan tan acotados que ya no es preciso un proceso de búsqueda para encontrar la solución $x_1 = 10$, $x_2 = 9$, $x_3 = 8$. Por ello, se remarca la utilidad de realizar un preproceso de k -consistencia previo al proceso de búsqueda.

■

10.2. El sudoku es un claro ejemplo de representación del puzzle como un problema de satisfacción de restricciones. Este problema se publicó en Nueva York en el año 1979 bajo el nombre de “Number Place” y se hizo popular en Japón con el nombre de sudoku (“*Sudji wa dokushin ni kagiru*”: “los números deben ser sencillos” o “los números deben aparecer una vez”). En el problema del sudoku (ver un ejemplo en la Figura 10.23), hay que llenar las casillas de un tablero de 9×9 con números del 1 al 9, de forma que no se repita ningún número en la misma fila, columna, o subcuadro de 3×3 que componen el sudoku. Modelar este problema como un CSP.

6	1	4	5
	8 3	5 6	
2			1
8	4	7	6
	6		3
7	9	1	4
5			2
	7 2	6 9	
4	5	8	7

Figura 10.23: Ejemplo de Sudoku.

Solución: Este problema se puede modelar como un CSP de forma muy sencilla. Cada celda del tablero se puede ver como una variable, cuyo dominio son los valores naturales entre el 1 y el 9. Por lo tanto, existirán 9 x 9 variables. La especificación del CSP es:

- **Variables:** $(x_{11}, x_{12}, \dots, x_{19}, x_{21}, x_{22}, \dots, x_{29}, \dots, x_{91}, x_{92}, \dots, x_{99})$
- **Dominios:** Para todas las variables, el dominio es: $\{1, 2, \dots, 9\}$
- **Restricciones:** Debe plantearse una restricción binaria de desigualdad entre:

– Todos los pares de variables en cada fila:

$$\neq (x_{11}, x_{21}, x_{31}, \dots, x_{91}), \neq (x_{12}, x_{22}, x_{32}, \dots, x_{92}), \dots, \neq (x_{19}, x_{29}, x_{39}, \dots, x_{99})$$

– Todos los pares de variables en cada columna:

$$\neq (x_{11}, x_{12}, \dots, x_{19}), \neq (x_{21}, x_{22}, \dots, x_{29}), \dots, \neq (x_{91}, x_{92}, \dots, x_{99})$$

– Y todos los pares de variables en cada submatriz.

$$\neq (x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}), \dots \text{ idem para las restantes submatrices.}$$

■

10.3. Cinco casas consecutivas tienen colores diferentes y son habitadas por hombres de diferentes nacionalidades. Cada uno tiene un animal diferente, una bebida preferida y fuma una marca determinada. Además, se sabe que:

1. *El noruego vive en la primera casa*
2. *La casa de al lado del noruego es azul*
3. *El habitante de la tercera casa bebe leche*
4. *El inglés vive en la casa roja*
5. *El habitante de la casa verde bebe café*
6. *El habitante de la casa amarilla fuma Kools*
7. *La casa blanca se encuentra justo después de la verde*
8. *El español tiene un perro*
9. *El ucraniano bebe té*
10. *El japonés fuma Cravens*
11. *El fumador de Old Golds tiene un caracol*
12. *El fumador de Gitanes bebe vino*
13. *El vecino del fumador de Chesterfields tiene un reno*
14. *El vecino del fumador de Kools tiene un caballo*

Solución: Este problema admite múltiples especificaciones. Podemos definir variables de la siguiente forma, por ejemplo, para la casa 1:

- Casa 1- nacionalidad $\in \{\text{noruego, inglés, ucraniano, japonés, español}\}$
- Casa 1-animal $\in \{\text{perro, caracol, reno, caballo, cebra}\}$
- Casa 1- fuma $\in \{\text{kools, cravens, golds, gitanes, chesterfields}\}$
- Casa 1-bebida $\in \{\text{leche, café, té, vino, agua}\}$
- Casa 1- color $\in \{\text{azul, roja, verde, amarilla, blanca}\}$

Y así sucesivamente para el resto de las casas. Sin embargo, esta definición de variables haría muy compleja la especificación de las restricciones. Otra definición más adecuada sería:

- **Variables:** azul, roja, verde, amarilla, blanca, noruego, inglés, ucraniano, japonés, español, perro, caracol, reno, caballo, cebra, leche, café, té, vino, agua; kools, cravens, golds, gitanes, chesterfields.
- **Dominios:** Todas las variables se instancian en el dominio $\{1, 2, 3, 4, 5\}$, indicando la casa correspondiente.

• **Restricciones:**

$$\begin{aligned}
 R_1: \text{noruego} &= 1; & R_2: \text{azul} &= \text{noruego} + 1 \vee \text{azul} + 1 &= \text{noruego}; \\
 R_3: \text{leche} &= 3; & R_4: \text{inglés} &= \text{roja}; & R_5: \text{verde} &= \text{café}; \\
 R_6: \text{amarilla} &= \text{kools}; & R_7: \text{blanca} &= \text{verde} + 1; & R_8: \text{español} &= \text{perro}; \\
 R_9: \text{ucraniano} &= \text{té}; & R_{10}: \text{japonés} &= \text{craven}; & R_{11}: \text{golds} &= \text{caracol}; \\
 R_{12}: \text{gitanes} &= \text{vino}; & R_{13}: \text{chesterfields} &= \text{reno} + 1 \vee \text{chesterfields} + 1 &= \text{reno}; \\
 R_{14}: \text{kools} &= \text{caballo} + 1 \vee \text{kools} + 1 &= \text{caballo};
 \end{aligned}$$

Y, adicionalmente:

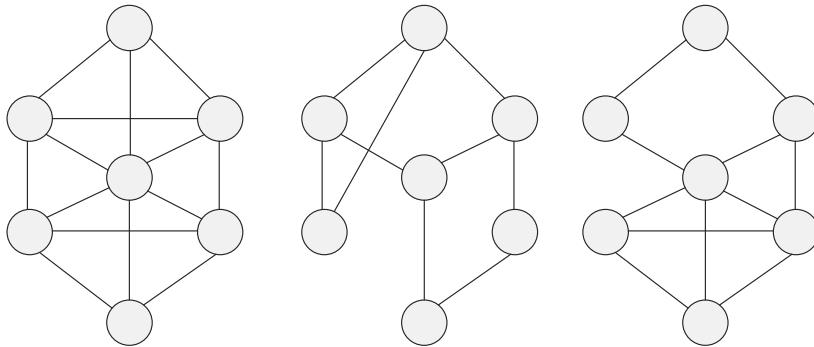
$$\begin{aligned}
 &\neq (\text{azul, roja, verde, amarilla, blanca}); \\
 &\neq (\text{noruego, inglés, ucraniano, japonés, español}); \\
 &\neq (\text{perro, caracol, reno, caballo, cebra}); \\
 &\neq (\text{leche, café, té, vino, agua}); \\
 &\neq (\text{kools, cravens, golds, gitanes, chesterfields});
 \end{aligned}$$

Con esta modelización de variables, ha sido fácil especificar las restricciones. Una buena especificación del CSP es importante respecto a la simplicidad de la representación, así como a la eficiencia en la resolución del problema.

■

10.10 Ejercicios propuestos

10.1. Obtener la ordenación de variables según los diferentes métodos de ordenación estáticos vistos en la sección 10.5.1 para cada uno de los CSP binarios siguientes:



10.2. En un pueblo viven 4 familias A, B, C y D en casas próximas cuyos números son: 1, 2, 3 y 4. D vive en una casa con menor número que B. B vive próximo a A en una casa con mayor número. Hay al menos una casa entre B y C. D no vive en una casa cuyo número es 2. C no vive en una casa cuyo número es 4. Modele el problema como un CSP, aplique diversos niveles de consistencia, y resuelva qué familia vive en cada casa aplicando Forward-Checking y Full Look-Ahead.

10.3. Un cuadrado mágico es la disposición de una serie de números enteros en un cuadrado o matriz de tal forma que la suma de los números por columnas, filas y diagonales sea la misma, la constante mágica. Usualmente los números empleados para rellenar las casillas son consecutivos, de 1 a n^2 , siendo n el número de columnas y filas del cuadrado mágico (en el ejemplo, $n = 3$). Hacer un CSP que obtenga cuadrados mágicos con diversos valores de n .

$$\text{Número Mágico}(n) = \frac{n(n^2+1)}{2}$$

4	9	2
3	5	7
8	1	6

10.4. Seis amigos y sus respectivas esposas, se hospedan en el mismo hotel; y todos ellos salen todos los días, asistiendo a reuniones de distinto volumen y composición. Para asegurar la variedad en estas salidas, han acordado establecer las siguientes reglas: “Si Antonio está con su mujer, es decir en la misma reunión que su mujer, y David con la suya, y Luis con la señora de Pedro, Enrique debe estar con la señora de Ramón. Si Antonio está con su mujer y Pedro con la suya, y David con la señora de Enrique, Ramón no debe estar con la señora de Luis. Si Enrique y Ramón y sus mujeres están todos en la misma reunión, y Antonio no está con la señora de David, Luis no debe estar con la señora de Pedro. Si Antonio está con su mujer y Ramón

con la suya, y David no está con la señora de Enrique, Luis debe estar con la señora de Pedro. Si Luis está con su mujer y Pedro con la suya y Enrique con la señora de Ramón, Antonio no debe estar con la señora de David. Si David y Enrique y sus mujeres están todos en la misma reunión, y Luis no está con la señora de Pedro, Ramón debe estar con la señora de Luis". Modelar dicho problema como un CSP y obtener las posibles combinaciones de reuniones para cada amigo. ¿Es posible que todos los días haya al menos un matrimonio cuyos miembros no estén juntos en la misma reunión? Este problema y el siguiente son originales de Lewis Carroll, en su obra Lógica Simbólica.

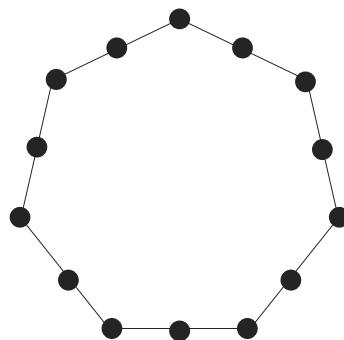
10.5. Cinco amigos, Bernardo, Casimir, Luis, Carlos y Marcial, se encuentran cada día en el restaurante. Tienen estas reglas, que observan cada vez que comen: "Bernardo toma sal si y solamente si Casimir toma sólo sal o solo mostaza. Bernardo toma mostaza si y solamente si, o bien Luis no toma sal ni mostaza, o bien Marcial toma ambas. Casimir toma sal si y solamente si, o Bernardo toma solamente uno de los dos condimentos, o bien Marcial no toma ninguno de ellos. Toma mostaza si y solamente si Luis o Carlos toman dos condimentos. Luis toma sal si y solamente si o bien Bernardo no toma ningún condimento, o bien Casimir toma ambos. Luis toma mostaza si y solamente si Carlos o Marcial no toman ni sal ni mostaza. Carlos toma sal si y solamente si Bernardo o Luis no toman ni sal ni mostaza. Toma mostaza si y solamente si Casimir o Marcial no toman ni sal, ni mostaza. Marcial toma sal si y solamente si Bernardo o Carlos toman los dos condimentos. Marcial toma mostaza si y solamente si Casimir o Luis toman sólo un condimento". Modelar el CSP correspondiente. El problema consiste en descubrir si estas reglas son compatibles y, en caso de que lo sean, cuáles son las combinaciones posibles.

10.6. Representar mediante un CSP la siguiente información: "Juana, Pepa y Paloma nacieron y viven en ciudades diferentes (Málaga, Madrid y Valencia). Además, ninguna vive en la ciudad donde nació. Juana es más alta que la que vive en Madrid. Paloma es cuñada de la que vive en Valencia. La que vive en Madrid y la que nació en Málaga tienen nombres que comienzan por distinta letra. La que nació en Málaga y la que vive ahora en Valencia tienen nombres que comienzan por la misma letra". ¿Dónde nació y vive cada una?

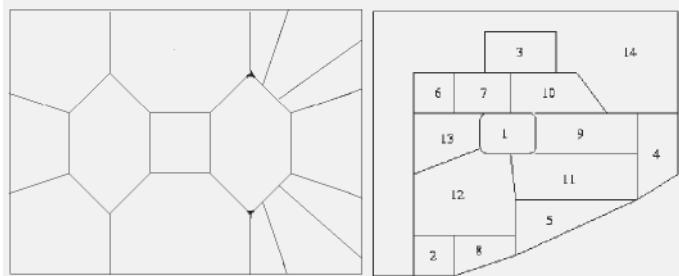
10.7. Obtener el CSP correspondiente al siguiente ejemplo de scheduling: Existen 3 periódicos (P_1, P_2, P_3) y 4 lectores (L_1, L_2, L_3, L_4), que desean leer los periódicos en el mismo orden y con la duración indicada en (P_1, P_2, P_3). Todos deben empezar a partir del ready-time y acabar antes del due-time, según la tabla siguiente:

	Ready-Time	P_1	P_2	P_3	Due-Time
L_1	0	5'	10'	2'	30'
L_1	0	2'	6'	5'	20'
L_3	0	10'	15'	15'	60'
L_4	0	3'	5'	5'	15'

10.8. Henry Dudeney (1847-1930) era un ingenioso inventor de problemas matemáticos. Entre sus aportaciones se encuentra un puzzle con cierta complejidad de resolución. Se trata de un heptágono donde en cada arista hay que colocar tres números: uno en cada vértice y otro en el centro de la arista. Hay que colocar los números del 1 al 14 alrededor de las aristas del heptágono de manera que el número de la arista y los dos vértices extremos sumen lo mismo, para todas las aristas. Modelar el CSP correspondiente.



10.9. Obtener, según las heurísticas de ordenación de variables vistas en la sección 10.5.1, la mejor ordenación para el coloreado de los siguientes mapas:



Referencias

- ALLEN, J.: «Maintaining knowledge about temporal intervals». *Comm. Of the ACM*, 1983, **26**(1), pp. 832–843.
- APT, K: *Principles of constraint programming*. Cambridge University Press, 2003. ISBN 0-521-82583-0.
- BACCHUS, FAHIEM: «Extending Forward Checking.» En: *CP*, pp. 35–51, 2000.
- BARBER, F.: «Monografía: Problemas de Satisfacción de Restricciones». *Revista Iberoamericana de Inteligencia Artificial*, 2003, **20**.
- BARTÁK, R.: «Theory and Practice of Constraint Propagation». En: *Proc. of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC2001), June 2001*, pp. 7–14, 2001.
- DECHTER, R.: *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
- DECHTER, R.; MEIRI, I. y PEARL, J.: «Temporal constraint networks». *Artificial Intelligence*, 1991, pp. 61–95.
- DECHTER, RINA y DECHTER, AVI: «Belief maintenance in dynamic constraint networks». En: *Proc.of AAAI-88 (St. Paul, MN)*, pp. 37–42, 1988.
- DENT, MICHAEL J. y MERCER, ROBERT E.: «Minimal Forward Checking.» En: *ICTAI*, pp. 432–438, 1994.
- DUBOIS, D.; FARGIER, H. y PRADE, H.: «The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction». En: *IEEE Int. Conf. on Fuzzy Systems*, pp. 1131–1136, 1993.
- FALTINGS, B. y YOKOO, M.: «Distributed Constraint Satisfaction». *Special Issue on Artificial Intelligence*, 2005, **161**(1-2).
- FREUDER, E. C.: «A sufficient condition for backtrack-free search». *Journal of the ACM*, 1982, **29**(1), pp. 24–32.
- FROST, DANIEL y DECHTER, RINA: «Dead-End Driven Learning». En: *Proceedings of the Twelfth National Conference of Artificial Intelligence(AAAI-94)*, volumen 1, pp. 294–300. AAAI Press, Seattle, Washington, USA. ISBN 0-262-61102-3, 1994. citeseer.ist.psu.edu/frost94deadend.html
- FRÜHWIRTH, THOM y ABDENNADHER, SLIM: *Essentials of constraint programming*. Springer, 2003. ISBN 3-540-67623-6.
- GASCHNIG, J.: «Performance measurement and analysis of certain search algorithms». *Informe técnico CMU-CS-79-124*, Carnegie-Mellon University, 1979.
- HENTENRYCK, P. V.: *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.

- MACKWORTH, A. K.: «Consistency in networks of relations». *Artificial Intelligence*, 1977, **8**, pp. 121–118.
- MARRIOT, KIM y STUCKEY, PETER J.: *Programming with Constraints*. MIT Press, 1998. ISBN 0-262-13341-5.
- MONTANARI, U.: «Networks of constraints: fundamental properties and applications to picture processing». *Information Science*, 1974, **7**, pp. 95–132.
- PROSSER, P.: «Hybrid algorithms for the constraint satisfaction problem». *Computational Intelligence*, 1993, **9(3)**, pp. 268–299.
- SALIDO, M. A. y BARBER, F.: «Distributed CSPs by Graph Partitioning». *Applied Mathematics and Computation*, 2006, **183(1)**, pp. 491–498.
- SCHIEIX, T.; FARGIER, H. y VERFAILLE, G.: «Valued constraint satisfaction Problems». En: *IJCAI-95*, pp. 631–637, 1995.
- STERGIOU, K. y WALSH, T.: «Encoding of non-binary constraints satisfaction problems». En: *In Proc. of the National Conference on Artificial Intelligence (AAAI-99)*, pp. 163–168, 1999.
- VAN BEEK, P.: «Temporal query processing with indefinite information». *Artificial Intelligence in Medicine*, 1991, **3(6)**, pp. 325–339.
- VILAIN, M. y KAUTZ, H.: «Constraint propagation algorithms for temporal reasoning». En: *Proc. of the AAAI-86*, pp. 377–382, 1986.
- WALLACE, G.: «Survey: Practical Applications of Constraint Programming». *Constraints Journal*, 1995, **1(1)**, pp. 139–168.
- YOKOO, M.; DURFEE, E.H.; ISHIDA, T. y KUWABARA, K.: «The distributed constraint satisfaction problem: formalization and algorithms». *IEEE Transactions on Knowledge and Data Engineering*, 1998, **10(5)**, pp. 673–685.

Capítulo 11

Computación Evolutiva

Fernando Jiménez Barrionuevo
Universidad de Murcia

11.1 Introducción

La *Computación Evolutiva* (CE) usa modelos computacionales de procesos evolutivos como elementos clave en el diseño e implementación de sistemas que resuelven problemas basados en una computadora. Existe una gran variedad de modelos de CE los cuales han sido propuestos y estudiados en las últimas décadas, y que podemos referenciarlos comúnmente como *Algoritmos Evolutivos* (AE). Desde el punto de vista de la *Algoritmia*, los AE son algoritmos probabilistas, es decir, toman decisiones aleatorias y se caracterizan fundamentalmente porque el mismo algoritmo puede comportarse de distinta forma aplicado a los mismos datos. Desde el punto de vista de la IA, los AE son algoritmos heurísticos, es decir, permiten encontrar buenas soluciones en tiempos razonables mediante la evaluación del progreso logrado en la búsqueda de un resultado final. La heurística incorporada en un AE se inspira en la evolución de los seres vivos, retomando los conceptos de selección natural y genética. Así pues, la CE interpreta la naturaleza como una inmensa máquina de resolver problemas y trata de encontrar el origen de dicha potencialidad para utilizarla en nuestros programas.

La CE surge a finales de los años 60 cuando John Holland se planteó la posibilidad de incorporar los mecanismos naturales de selección y supervivencia para resolver una gran cantidad de problemas de IA que ya habían sido “resueltos” muy eficientemente por la naturaleza pero que resultaban decepcionantemente inabordables mediante computadoras. El fruto de sus investigaciones se plasmó en el libro *Adaptation in Natural and Systems* [Holland, 1975], que con el paso del tiempo ha sido considerado como el punto de arranque de la CE. Los intereses de Holland y sus colaboradores fueron en principio académicos, tratando esencialmente de introducir un marco más amplio en el que englobar la IA con la intención de resolver ciertos problemas genéricos que constantemente aparecían en dicha disciplina. Sin embargo, a la hora de llevarlas a la práctica, las ideas de Holland resultaban, cuando menos, poco eficientes.

A mediados de los 80, la aparición de computadores de altas prestaciones y bajo coste cambia por completo el panorama. La CE se puede utilizar para resolver con

éxito ciertos tipos de problemas de la ingeniería y de las ciencias sociales que anteriormente eran difíciles de tratar. De esta época son los libros *Genetic Algorithms in Search, Optimization and Machine Learning* [Goldberg, 1989a], y *Handbook of Genetic Algorithms* [Davis, 1991]. En ellos se plantean y resuelven problemas “de la vida real” con los que se empiezan a suscitar intereses económicos.

Así pues, los AE se consolidan con el tiempo como potentes técnicas de búsqueda y optimización, permitiendo asimismo resolver gran cantidad de problemas propios de los procesos de aprendizaje. La clave de su poder reside en que reúnen un conjunto de características que cualquier técnica de búsqueda desearía tener. Los AE son robustos y están concebidos como métodos altamente no lineales de búsqueda global, capaces de resolver problemas de gran dimensionalidad, multimodales, no lineales, en presencia de ruido y dependientes del tiempo, lo que permite tratar problemas muy difíciles si no irresolubles. La búsqueda basada en poblaciones propia de los AE es su característica clave, ya que los sitúa como algoritmos intrínsecamente paralelos, lo cual, junto con los conceptos de selección, herencia y mutación, sienta las bases teóricas del buen funcionamiento de éstos, sintetizadas en el *Teorema del Esquema* [Michalewicz, 1992], el cual básicamente viene a decir que las buenas soluciones exploradas por un AE reciben un incremento exponencial de sus representantes en las subsecuentes generaciones.

Sin embargo, los AE no están exentos de inconvenientes. Fenómenos tales como la *epistasis*, la *decepción*, o el *genetic drift* [Goldberg, 1989a], hacen que la búsqueda caiga en óptimos locales, o impiden una localización diversificada de soluciones en problemas que lo requieren, aspectos estos que están siendo objeto de intensivos estudios por los investigadores en los últimos años.

Los AE presentan además unas características excelentes de cara a su diseño. Un AE no requiere muchos conocimientos matemáticos del problema para el cual está siendo diseñado. Debido a su naturaleza evolutiva, el AE busca soluciones sin considerar un conocimiento específico del problema, es decir, son métodos *débiles*. No obstante, un AE proporciona una gran flexibilidad para hibridizarse con conocimiento dependiente del problema, lo cual puede mejorar la eficiencia de la técnica en un problema concreto. Un AE puede implementar cualquier clase de función o funciones objetivo y/o restricciones, así como representar cualquier conjunto de variables de decisión definidas en espacios continuos, discretos, o híbridos.

Un AE típicamente mantiene una *población* de soluciones potenciales (*individuos*) del problema, las cuales evolucionan de acuerdo a reglas de selección y otros operadores, tales como recombinación y mutación. Cada individuo en la población es evaluado, recibiendo una medida de la adecuación (*fitness*) en su entorno. La selección enfoca su atención en los individuos de adecuación alta, explotando así la información de las adecuaciones disponibles. La recombinación y mutación modifican los individuos, proporcionando una heurística general de exploración. Aunque desde un punto de vista biológico estos algoritmos son extremadamente simples, proporcionan sin embargo mecanismos de búsqueda adaptativos muy potentes y robustos. El Algoritmo 11.1 muestra la estructura de un AE. El algoritmo comienza inicializando una población, normalmente de forma aleatoria, cuyos individuos son evaluados mediante una función de adecuación que usualmente coincide con la función objetivo del pro-

blema de optimización que se quiere resolver. El AE entra en un proceso iterativo que refleja el transcurso de la generaciones. En cada generación se realiza un proceso de selección de los mejores individuos (más adaptados, es decir, con mejor adecuación) los cuales se constituyen como padres para una recombinación y mutación, dando lugar a un nuevo conjunto de individuos $C(t)$ que son asimismo evaluados. Una nueva población es generada mediante un proceso de selección de supervivientes realizado entre la población actual $P(t)$ y los nuevos individuos del conjunto $C(t)$, produciendo una nueva población $P(t + 1)$ y una nueva generación. El proceso iterativo termina cuando se cumple una condición de terminación que normalmente se establece como la conclusión de un número prefijado de generaciones.

Algoritmo 11.1 Algoritmo evolutivo.

```

1:  $t \leftarrow 0$ ; /* generación inicial */
2: inicializar  $P(t)$ ;
3: evaluar  $P(t)$ ;
4: mientras (no se cumpla la condición de terminación) hacer
5:   seleccionar padres de  $P(t)$ ;
6:   recombinar padres y mutar  $\Rightarrow C(t)$ ;
7:   evaluar  $C(t)$ ;
8:   seleccionar supervivientes de  $P(t) \cup C(t) \Rightarrow P(t + 1)$ ; /* sustitución generacional */
9:    $t \leftarrow t + 1$ ; /* siguiente generación */
10: fin mientras
```

Generalmente se acepta que un AE debe contener los siguientes cinco componentes:

- Una *representación* apropiada de las soluciones del problema. Este punto es muy importante en el diseño de un AE ya que establece el espacio de búsqueda y de su elección depende la eficiencia del algoritmo. En las siguientes secciones se verán representaciones binarias propias de los Algoritmos Genéticos, así como representaciones *ad-hoc* típicas de determinados problemas de optimización.
- Una forma de crear una *población inicial* de soluciones. Aunque usualmente la población inicial se genera de forma aleatoria dentro del espacio de búsqueda, la incorporación de conocimiento puede ayudar a guiar la búsqueda.
- Una *función de evaluación* capaz de medir la adecuación de cualquier solución, y que hará el papel de “entorno”, en el cual las mejores soluciones, esto es, aquellas con mejor adecuación, tengan mayor probabilidad de selección y supervivencia.
- Un conjunto de *operadores evolutivos* que actúan como reglas de transición probabilísticas (no deterministas) para guiar la búsqueda, y que combinan entre sí las soluciones existentes con el propósito de obtener otras nuevas.
- El valor de unos *parámetros* de entrada que el AE usa para guiar su evolución (tamaño de la población, número de iteraciones, probabilidades de aplicación de los operadores evolutivos, etc.).

Aunque la CE constituye una gran familia de técnicas entre las que se encuentran las *Estrategias de Evolución* [Rechenberg, 1973], la *Programación Evolutiva* [Fogel y

otros, 1966], la *Programación Genética* [Koza, 1972] y los *Algoritmos Genéticos* [Goldberg, 1989a], son estos últimos los que han recibido más atención y han estado sujetos a un mayor estudio tanto teórico como aplicado, por lo que nos dedicaremos a ellos fundamentalmente en este capítulo. La sección 11.2 describe los elementos fundamentales de un algoritmo genético simple, ilustrándose con un ejemplo de optimización sencillo. La sección 11.4 muestra algunas técnicas avanzadas usadas en la literatura de cara al diseño eficiente de AE en ambientes de optimización complejos, entre los que se incluyen la presencia de restricciones o la optimización multiobjetivo. Se incluyen también algunas consideraciones generales de cara a la evaluación y validación requeridas en el diseño e implementación de AE. La sección 11.7 muestra ejemplos de AE para problemas conocidos como el del viajante de comercio y el de selección de variables en Minería de Datos. Finalmente la sección 11.8 es una propuesta de ejercicio para el diseño, implementación y evaluación de distintos AE para el pasatiempo del Sudoku.

11.2 Un algoritmo genético simple

Los Algoritmos Genéticos (AG) fueron introducidos inicialmente por Holland en 1975 para abstraer y explicar rigurosamente los procesos adaptativos de los sistemas naturales, así como para el diseño de sistemas artificiales de software que retengan los mecanismos importantes de los sistemas naturales. Fue unos años más tarde cuando su alumno, D. Goldberg, implementó el primer AG aplicado en problemas industriales. Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron los AGs en un campo suficientemente aceptado para celebrar la primera conferencia en 1985, ICGA '85, la cual se sigue celebrando, actualmente integrada dentro de la Genetic and Evolutionary Computation Conference (GECCO).

Los AG han usado tradicionalmente una representación más independiente del dominio, normalmente, cadenas binarias. Sin embargo, muchas aplicaciones recientes de AG han usado otras representaciones, tales como grafos, expresiones Lisp, listas ordenadas, o vectores de parámetros reales. En esta sección describiremos un AG simple que usa como representación cadenas de bits. El AG que mostramos¹ es el descrito en los trabajos iniciales de Goldberg el cual supuso la lanzadera en el campo de la Computación Evolutiva, tanto a nivel de aplicaciones prácticas, como a nivel teórico, sentando las bases de convergencia al óptimo mediante el Teorema del Esquema.

11.2.1 Representación

En un AG, las soluciones potenciales al problema se representan normalmente mediante cadenas binarias de bits (0's y 1's) de una longitud determinada *long* que vendrá impuesta por el número de variables existentes en la solución y por el número de bits necesarios para codificarlas. Otros términos usados a menudo para denominar una solución del problema en un AG son *string* o *estructura*, y, siguiendo el vocabulario

¹En [Goldberg, 1989a], Apéndice C, puede encontrarse el código completo en Pascal de un AG simple.

de los sistemas biológicos, *cromosoma*. Así, los cromosomas están compuestos por unidades binarias que se denominan *genes*. Al valor de un gen determinado se le denomina *alelo*, y a su posición en el cromosoma *locus*. Al paquete genético total se le denomina *genotipo*, y a la interacción del genotipo con su entorno se le denomina *fenotipo*, que se traduce en la decodificación del cromosoma para la obtención de una solución alternativa (conjunto de parámetros particular, o un punto en el espacio de búsqueda). De esta forma, podemos representar un cromosoma c_i^t en una *generación* (iteración) determinada t como:

$$c_i^t = (b_{i1}^t \dots b_{ilong}^t)$$

con $b_{ij}^t \in \{0, 1\}$, $j = 1, \dots, long$. El término *individuo* es frecuentemente utilizado [Goldberg, 1989a] para referirse al conjunto de información *genotipo-fenotipo-adecuación*. Así, podemos representar un individuo X_i^t en una generación t , como la terna:

$$X_i^t = (c_i^t, x_i^t, f_i^t)$$

donde x_i^t es la decodificación (fenotipo) del cromosoma c_i^t , y f_i^t es la adecuación de la solución al entorno o *fitness*.

11.2.2 Obtención de la población inicial

Una población en un AG está formada por un conjunto de m individuos, donde el número m (tamaño de la población) es un parámetro de entrada al AG. De esta forma, en una generación determinada t , podemos representar una población $P(t)$ como:

$$P(t) = \{X_1^t, \dots, X_m^t\}$$

Para obtener una población inicial $P(0) = \{X_1^0, \dots, X_m^0\}$ debemos generar m individuos iniciales. Un procedimiento de inicialización de un individuo X_i^0 consiste simplemente en asignar, para cada gen b_{ij}^0 , $j = 1, \dots, long$ de su cromosoma c_i^0 , un valor aleatorio 0 ó 1.

Con la decodificación del cromosoma c_i^0 obtendremos su fenotipo x_i^0 , y la adecuación de la solución al entorno se obtiene a través de la función de evaluación.

11.2.3 Función de evaluación

Aunque existen otras alternativas, el mecanismo usual de medir la adecuación de una solución consiste simplemente en evaluar su fenotipo a través de la función objetivo f del problema que se esté resolviendo. Así pues, la función de evaluación $eval$ se corresponde normalmente con la función objetivo f del problema, y entonces, dado un cromosoma c_i^t y su fenotipo x_i^t , podemos obtener su adecuación f_i^t como:

$$f_i^t = eval(c_i^t) = f(x_i^t)$$

La función objetivo juega un papel fundamental en un AE, puesto que ésta es la información fundamental que se usa del entorno, lo cual hace que estos métodos sean muy generales y robustos.

11.2.4 Selección, muestreo, operadores genéticos y sustitución generacional

Una vez que se han evaluado todas las soluciones de la población en una generación, el proceso evoluciona hacia una nueva generación. La población en la nueva generación t sufrirá una transformación con respecto a la población en la generación anterior $t - 1$. Esta transformación se lleva a cabo, en un esquema básico de funcionamiento (véase la Figura 11.1), por medio de la aplicación de las siguientes reglas de transición probabilísticas:

1. Selección/muestreo de padres.
2. Cruce.
3. Mutación.
4. Selección de supervivientes/sustitución generacional.

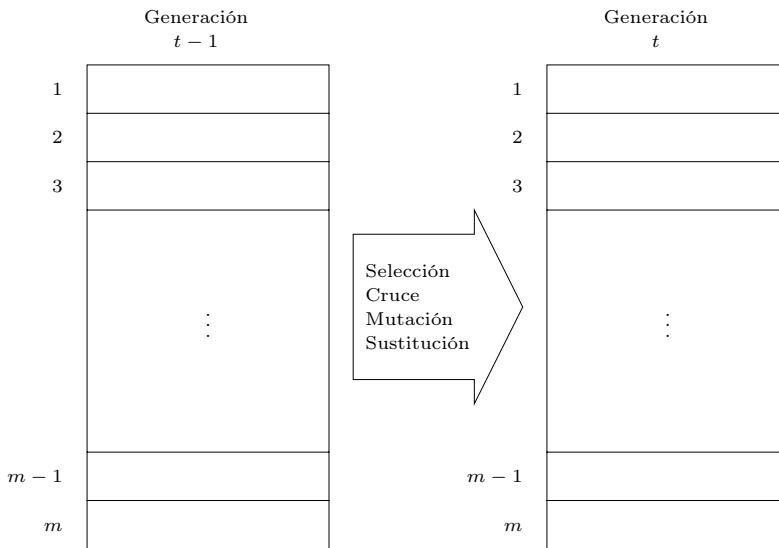


Figura 11.1: Transformación de la población en un AG.

Selección y muestreo El esquema de selección asigna, a cada individuo i de una población en la generación t , el número esperado de descendientes $N_i^t = m \cdot p_i^t$, donde p_i^t es la probabilidad de que el individuo i sea seleccionado. Asociado a un esquema de selección, un algoritmo de muestreo obtiene individuos de la población (números discretos) de acuerdo al número esperado de descendientes de los individuos. En el AG simple que estamos describiendo se usa, como esquema de selección, la

selección proporcional, y como algoritmo de muestreo, el *muestreo estocástico con reemplazamiento* [DeJong, 1975] o *rueda de ruleta* [Goldberg, 1989a].

Con la selección proporcional se asigna a cada cromosoma de una población en la generación t , la siguiente probabilidad:

$$p_i^t = \frac{f_i^t}{\sum_{i=1}^m f_i^t}, \quad i = 1, \dots, m.$$

El algoritmo de muestreo estocástico con reemplazamiento realiza los siguientes pasos:

- Calcular la probabilidad acumulada de cada cromosoma:

$$q_i^t = \sum_{j=1}^i p_j^t, \quad i = 1, \dots, m;$$

- Repetir m veces los siguientes pasos:

- Extraer, con reemplazamiento, un número real aleatorio $r \in [0, 1]$.
- Si $r \leq q_1$ entonces se muestrea el primer cromosoma de la población en la generación t , y en otro caso se muestrea el i -ésimo cromosoma ($2 \leq i \leq m$) tal que $q_{i-1}^t < r \leq q_i^t$.

Este proceso de muestreo representa una rueda de ruleta con m marcas y separaciones entre ellas fijadas proporcionalmente de acuerdo a la adecuación de los cromosomas.

Operadores genéticos Una vez seleccionados un par de padres, el *operador de cruce* se aplica a éstos con probabilidad p_{cruz} , donde p_{cruz} es un parámetro de entrada el cual nos determina el número esperado de cromosomas a los cuales se les aplicará el operador de cruce en cada generación. Con el *cruce simple* [Goldberg, 1989a], se genera un número entero aleatorio $pos \in [1..long - 1]$, que indica la posición del punto de cruce. Una vez seleccionado el punto de cruce y dados dos cromosomas, c_i^t y c_j^t , seleccionados como padres en la generación t :

$$c_i^t = (b_{i1}^t \dots b_{ipos}^t b_{i(pos+1)}^t \dots b_{ilong}^t)$$

$$c_j^t = (b_{j1}^t \dots b_{jpos}^t b_{j(pos+1)}^t \dots b_{jlong}^t)$$

el cruce simple generaría los siguientes *hijos*:

$$c'_i^t = (b_{i1}^t \dots b_{ipos}^t b_{j(pos+1)}^t \dots b_{jlong}^t)$$

$$c'_j^t = (b_{j1}^t \dots b_{jpos}^t b_{i(pos+1)}^t \dots b_{ilong}^t)$$

los cuales contienen información genética cruzada de ambos padres. Seguidamente se aplica, a cada hijo, el *operador de mutación*. En el AG que estamos describiendo se usa la *mutación simple*. En cada gen del cromosoma se lleva a cabo un cambio de 0 a 1, o viceversa, con probabilidad p_{mut} , parámetro éste de entrada que nos da el número esperado de genes mutados en cada generación.

Sustitución generacional Cuando tenemos un conjunto de descendientes producidos mediante selección-cruce-mutación, el siguiente paso es evaluarlos y decidir qué individuos constituirán la nueva población. Los individuos de la nueva población se elegirán, en un esquema general como el que muestra el Algoritmo 11.1, de entre la población actual y los descendientes.

En el AG simple que estamos describiendo se usa el esquema de *sustitución generacional completa*, junto con una estrategia elitista. Con el elitismo, el mejor individuo de la población actual sobrevive en la nueva población. El resto de individuos de la nueva población lo constituyen los descendientes generados vía selección-cruce-mutación, hasta formar una nueva población completa de m individuos. Obviamente, los individuos con mayor probabilidad de selección (mayor adecuación) se seleccionarán un número mayor de veces, y tendrán mayor contribución en las subsiguientes generaciones.

El resto de la evolución, como muestra el Algoritmo 11.1, es simplemente un proceso iterativo de selección-cruce-mutación-evaluación-sustitución en el transcurso de las generaciones hasta que se cumple la condición de parada, que normalmente consiste en alcanzar un número determinado de generaciones. Otras condiciones de parada consisten en el cumplimiento de algún predicado de éxito cuando es posible reconocer soluciones optimales o, en su defecto, satisfactorias para un decisor.

11.2.5 Parámetros de entrada

En cuanto a los parámetros que el AG utiliza, los más significativos son el *tamaño de la población*, el *número de generaciones*, la *probabilidad de cruce* y la *probabilidad de mutación*. De Jong [DeJong, 1975], completando los primeros estudios de Holland, realiza un análisis para la correcta fijación de parámetros haciendo distinción entre situaciones *off-line* y situaciones *on-line*. Para el tamaño de la población, tales estudios apuntan a un buen rendimiento off-line para poblaciones grandes, ya que convergen finalmente mejor debido a una mayor diversidad, y a un buen rendimiento on-line para poblaciones pequeñas, ya que éstas tienen la habilidad de cambiar más rápidamente y ofrecer una convergencia inicial mejor. En [Goldberg, 1989b] se establece una guía para la elección de posibles tamaños de población para representaciones binarias de longitud fija en función de la longitud de los cromosomas. Asimismo, en [Goldberg, 1989b] se estima que el número esperado de generaciones hasta la convergencia es una función logarítmica del tamaño de la población. Para las probabilidades de cruce y mutación, los estudios realizados en el tema apuntan a probabilidades de cruce altas y probabilidades de mutación bajas. Los siguientes valores han sido considerados como aceptables para un buen rendimiento de los AG (representación binaria) para funciones de optimización:

- Tamaño de la población: 50–100.
 - Probabilidad de cruce: 0.60.
 - Probabilidad de mutación 0.001.

11.2.6 Ejemplo: Optimización de una función simple

En esta sección consideraremos la optimización de una función simple de una variable para ilustrar el funcionamiento del AG simple. Dada la siguiente función [Michalewicz, 1992]:

$$f(x) = x \cdot \sin(10\pi \cdot x) + 1,0$$

el problema de optimización consiste en encontrar $x \in [-1, 2]$ que maximice la función f , es decir, encontrar x^* tal que:

$$f(x^*) > f(x), \quad \forall x \in [-1, 2].$$

Representación Asumimos que se requiere una precisión de 6 dígitos después del punto decimal. Puesto que el dominio de la variable x tiene longitud 3, la precisión requerida implica que el dominio $[-1, 2]$ debe ser dividido en $3 \cdot 10^6 = 3000000$ partes de igual tamaño, lo que significa que se requieren 22 bits para representar un cromosoma ($long = 22$), ya que:

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304.$$

Para decodificar un cromosoma $c_i^t = (b_{i1}^t \dots b_{i22}^t)$ realizamos los dos siguientes pasos:

1. Calcular la representación en base 10 del cromosoma:

$$x'_i = \sum_{j=1}^{22} b_{ij}^t \cdot 10^{22-j};$$

2. Realizar un cambio de escala de acuerdo a los valores límite del dominio de la variable. El fenotipo x_i^t resultante es:

$$x_i^t = -1, 0 + x'_i \cdot \frac{3}{2^{22} - 1}$$

donde -1 es el límite inferior del dominio, y 3 la longitud del dominio.

Los cromosomas 00000000000000000000000000 y 11111111111111111111111111 representan los valores -1,0 y 2,0 respectivamente.

Población inicial y evaluación La población inicial se puede obtener fácilmente generando aleatoriamente m cromosomas de 22 bits, para seguidamente proceder a sus decodificaciones y evaluaciones. La evaluación de un cromosoma c_i^t se obtiene para este ejemplo de la siguiente forma:

$$\text{eval}(c_i^t) = f(x_i^t) = x_i^t \cdot \sin(10\pi \cdot x_i^t) + 1,0 = f_i^t$$

La Tabla 11.1 muestra un ejemplo de 3 individuos (cromosoma, fenotipo, adecuación).

Cromosoma c_i^t	Fenotipo x_i^t	Adecuación f_i^t
1000101110110101000111	0,637197	1,586345
0000001110000000010000	-0,958973	0,078878
1110000000111111000101	1,627888	2,250650

Tabla 11.1: Tres individuos de longitud 22 para el problema de optimización del ejemplo.

Cruce Supongamos que los cromosomas 000000111000000010000 y 1110000000111 111000101 se han seleccionado como padres. El operador de cruce se aplica establecido un punto de cruce en la sexta posición. La Figura 11.2 muestra los hijos resultantes, con las siguientes evaluaciones:

$$\begin{aligned} \text{eval}(000000000011111000101) &= f(-0,998113) = 0,940865 \\ \text{eval}(111000111000000010000) &= f(1,666028) = 2,459245 \end{aligned}$$

Nótese que el segundo hijo tiene mejor adecuación que sus dos padres.

00000		01110000000010000	\Rightarrow	00000		0000011111000101
11100		0000011111000101	\Rightarrow	11100		01110000000010000

Figura 11.2: Cruce simple en la posición 6.

Mutación: Supongamos que operador de mutación se aplica al cromosoma 1110000 00011111000101. Dada la baja probabilidad de mutación, el operador aplica el cambio únicamente en el gen 10 del cromosoma. La Figura 11.3 muestra el descendiente. La evaluación del descendiente es:

$$\text{eval}(111000000111111000101) = f(1,630818) = 2,343555$$

lo que significa que el individuo ha mejorado tras la mutación.

1110000000111111000101	\Rightarrow	1110000001111111000101
------------------------	---------------	------------------------

Figura 11.3: Mutación en el gen 10.

Resultados experimentales El mejor cromosoma obtenido después de 150 generaciones, con $m = 50$, $p_{cruz} = 0,25$ y $p_{mut} = 0,01$ ha sido:

$$eval(1111001101000100000101) = f(1,850773) = 2,850227.$$

11.3 Fundamentos de los algoritmos genéticos

¿Por qué trabajan bien los AG? Para responder esta pregunta tendríamos que recurrir a todo el formalismo teórico que sustenta este tipo de algoritmos [DeJong, 1975; Goldberg, 1989a; Holland, 1975; Michalewicz, 1992]. Sin intención de utilizar de forma exhaustiva y rigurosa tal formalismo en un capítulo introductorio como este, podemos, no obstante, realizar algunos comentarios de carácter general sobre los fundamentos de los AG.

En un AG, la búsqueda de soluciones idóneas de un problema consiste en la búsqueda de determinadas cadenas binarias, de forma que el universo de todas las posibles cadenas puede ser concebido como un paisaje imaginario con cimas y valles que albergan buenas y malas soluciones respectivamente. También podemos definir *regiones* en el espacio de soluciones fijándonos en las cadenas que posean unos o ceros en lugares determinados, y que usualmente son denominadas *esquemas* o *hiperplanos*.

Un esquema es por tanto un patrón de similitud que describe un subconjunto de cadenas con similitudes en ciertas posiciones de éstas, y se define sobre el alfabeto ternario $\{0, 1, *\}$, donde el símbolo $*$ en una posición del esquema indica que el alelo en dicha posición no está determinado. Así, por ejemplo, para cadenas de longitud 5, el esquema $(1 * * * *)$ representa todas las cadenas que comienzan por 1 en el espacio de posibilidades, y el esquema $(1 * 0 * 1)$ describe un subconjunto de cuatro miembros $\{(10001), (10011), (11001), (11011)\}$.

De esta forma, el conjunto de cadenas que forma una población en el AG sondea el espacio de soluciones en muchos esquemas a la vez. Por tanto, una de las claves de la buena conducta de los AG reside en que cada cadena individual pertenece a todos los esquemas en los cuales aparece uno cualquiera de sus bits. Un AG que manipule una población de un número determinado de cadenas está tomando muestras de un número de esquemas enormemente mayor.

J.H. Holland estimó que el número de esquemas procesados útilmente en un AG que manipule un población de m individuos es del orden de m^3 , resultado éste importante al que se dio el nombre de *paralelismo implícito*.

El AG explota los esquemas de más alto rendimiento del espacio de soluciones, ya que las sucesivas generaciones de reproducción y cruce generan un número creciente de cadenas pertenecientes a ellas. La mutación, que por sí sola no puede generar progresos en la búsqueda de una determinada solución, proporciona un mecanismo

de exploración (nuevos esquemas) que impide el desarrollo de una población uniforme e incapaz de ulterior evolución. Ahora bien, el efecto de los operadores de cruce y mutación puede provocar también que determinadas cadenas abandonen el esquema de sus progenitores en el transcurso de las generaciones.

La probabilidad de que una cadena particular abandone el esquema de sus progenitores depende del *orden* y de la *longitud* de dicho esquema. El orden $o(H)$ se define como el número de posiciones fijas del esquema H , y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto de la mutación. La longitud $\delta(H)$ se define como la distancia entre la primera posición fija y la última posición fija en el esquema H , y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto del cruce.

Así pues, los esquemas constituidos con pocas y contiguas posiciones fijas que además tengan una adecuación por encima de la media, y que llamamos *bloques constructivos*, tienen mayor probabilidad de salir intactos de cruces y mutaciones y propagarse a las generaciones futuras.

Aplicando únicamente el operador de reproducción, el número esperado de representantes del esquema H en la población en el instante $t+1$ viene dado por la siguiente expresión:

$$N(H, t+1) = N(H, t) \cdot \frac{f(H, t)}{\bar{f}^t}$$

donde $N(H, t)$ es el número de representantes del esquema H en la población en el instante t , $f(H, t)$ es la adecuación media de los representantes del esquema H en la población en el instante t , y $\bar{f}^t = \sum_{i=1}^m f_i^t / m$ es la adecuación media de la población en el instante t . Si asumimos que un esquema H permanece por encima de la media en un $\epsilon\%$, es decir, $f(H, t) = \bar{f}^t + \epsilon \cdot \bar{f}^t$, entonces:

$$N(H, t+1) = N(H, 0)(1 + \epsilon)^t$$

lo que significa que los esquemas por encima de la media reciben un incremento exponencial de sus representantes en las siguientes generaciones.

Si consideramos también el efecto de los operadores de cruce y mutación, tenemos que:

$$N(H, t+1) \geq N(H, t) \cdot \frac{f(H, t)}{\bar{f}^t} \left[1 - p_{cruz} \cdot \frac{\delta(H)}{\text{long} - 1} - o(H)p_{mut} \right]$$

Tal efecto no es significante si el esquema es de longitud corta y de orden bajo, por lo que en tales condiciones se sigue produciendo un incremento exponencial de representantes del esquema en las siguientes generaciones.

Para concluir, podemos resumir lo comentado anteriormente a través de la *Hipótesis del Bloque Constructivo* y del *Teorema Fundamental de los AG* o *Teorema del Esquema*:

Hipótesis del Bloque Constructivo:

Un AG busca soluciones cerca del óptimo a través de la yuxtaposición de esquemas de longitud corta, orden bajo y adecuación alta, llamados bloques constructivos.

Teorema del Esquema:

Los esquemas de longitud corta, orden bajo y adecuación por encima de la media reciben un incremento exponencial de sus representantes en sucesivas generaciones de un AG.

11.4 Diseño de algoritmos evolutivos

Aunque la teoría que sustenta a los AE proporciona una explicación de por qué estos algoritmos convergen en soluciones optimales, en aplicaciones prácticas surgen inconvenientes que hacen que no siempre se cumplan los resultados teóricos. Una de las causas comunes que hacen que los AE se vean incapacitados para encontrar soluciones óptimas en determinadas circunstancias es el fenómeno de la *convergencia prematura*. Tal problema, que aparece también en otras técnicas de optimización, consiste, como su nombre indica, en una convergencia demasiado rápida, posiblemente en un óptimo local. En CE, esto es debido a la presencia de *superindividuos* en la población, los cuales son mucho mejores que la adecuación media de la población, por lo que acarrean un gran número de descendientes e impiden que otros individuos contribuyan con su descendencia en la siguiente generación, con la consecuente pérdida de información, lo que lleva consigo la formación de poblaciones altamente uniformes e incapaces de evolucionar. No obstante, ciertos diseños de AE son más propensos a la convergencia prematura que otros, y los investigadores más conocidos en el tema aluden a los mecanismos de selección y muestreo, a la ruptura de esquemas debido al cruce, a la fijación de parámetros y a las características propias de la función. Desafortunadamente, la convergencia prematura no es el único problema con el que se encuentran los AE. Aunque los AE están clasificados como algoritmos *débiles*, existe una gran variedad de problemas, los cuales resultan difíciles o imposibles de resolver con el esquema simple de AG visto anteriormente como, por ejemplo, problemas que contienen restricciones no triviales, problemas con función multimodal, problemas con múltiples objetivos, etc. Todo esto ha llevado en los últimos años al estudio de mejoras y extensiones sobre los AE que proporcionan un mejor rendimiento de éstos, así como un mayor ámbito de aplicación, tales como representaciones alternativas, mecanismos de obtención de poblaciones iniciales, funciones de evaluación, esquemas de selección, muestreo y sustitución generacional, nuevos operadores de cruce y mutación, técnicas basadas en el conocimiento específico del problema, técnicas para el manejo de restricciones, técnicas de diversidad, optimización multiobjetivo, etc. [Goldberg, 1989a] [Michalewicz, 1992].

En esta sección describimos algunas de estas mejoras y extensiones, destacables por haber proporcionado buenos rendimientos y por su aplicabilidad en un gran número de entornos de actual relevancia. Destacamos la representación de soluciones, los esquemas de selección, muestreo y sustitución generacional, los operadores de variación, el manejo de restricciones, y la optimización multiobjetivo, como aspectos más significativos.

11.4.1 Representación

La elección de la representación de las soluciones del problema adquiere una gran importancia en CE. La representación elegida puede limitar directamente la forma en la que el sistema observa su ambiente, y repercute en gran medida en el diseño de otros componentes del sistema. Existe una gran variedad de alternativas de representación en CE. Tal variedad aparece incluso considerando únicamente alfabetos binarios, encontrándonos con representaciones de longitud fija y de longitud variable, representaciones con codificación binaria y con codificación Gray, representaciones haploides y diploides, etc. [Goldberg, 1989a]. El alfabeto binario facilita los análisis teóricos y permite el diseño de operadores genéticos elegantes. Sin embargo, la representación binaria puede presentar inconvenientes cuando se aplica a problemas numéricos multidimensionales que requieren una alta precisión, o en presencia de restricciones no triviales [Michalewicz, 1992], y puede resultar difícil de aplicar y no natural en muchos problemas prácticos. Por otro lado, el paralelismo implícito no depende del uso de representaciones binarias, por lo que se abre un camino hacia el uso de representaciones distintas. Estas representaciones pueden diferir de la representación con cadenas binarias tanto en la cardinalidad del alfabeto, como en la estructura propiamente dicha. Así, se han considerado alfabetos de mayor cardinalidad, como pueden ser los enteros y los reales, que resultan especialmente útiles en problemas de optimización numérica, y una gran variedad de estructuras de datos diferentes como, por ejemplo, matrices, permutaciones, listas, árboles, grafos, etc. Aunque la implicación de la representación en el rendimiento de los AG es un tema que sigue aún bastante inexplorado, muchos investigadores [Michalewicz, 1992] mantienen que una representación natural de las soluciones del problema junto con una familia de operadores genéticos aplicables, puede ser bastante útil en la aproximación de soluciones para muchos problemas.

11.4.2 Esquemas de selección, muestreo y sustitución generacional

En el AG simple descrito en la sección anterior se han utilizado la selección proporcional, el muestreo estocástico con reemplazamiento, y la sustitución generacional completa. Con la selección proporcional se tiene el inconveniente de que la presencia de individuos por encima de la adecuación media puede dar lugar a la convergencia prematura. Con el muestreo estocástico con reemplazamiento puede ocurrir que un mismo individuo sea seleccionado un número excesivo de veces (incluso podría completar la nueva población), lo que daría lugar a poblaciones muy uniformes. La sustitución generacional completa puede producir un efecto negativo en la componente de explotación del AG. Todos estos inconvenientes motivaron el estudio de mejoras en el diseño de AG cuyos primeros y más reconocidos trabajos fueron realizados por DeJong [DeJong, 1975]. Aquí describiremos algunas de las técnicas que han tenido un mayor impacto: selección por ranking, selección por torneo, muestreo universal, sustitución steady-state, y el modelo del factor de crowding.

Selección por ranking Con estos métodos, los individuos son ordenados en una lista de acuerdo a sus adecuaciones, y la probabilidad de selección de un individuo se obtiene según su posición en la lista ordenada. Así, con el *ranking lineal* la probabilidad de selección de un cromosoma c_i^t se obtiene como:

$$p_i^t = (a_{max} - (a_{max} - a_{min})) \cdot \frac{rank(c_i^t) - 1}{m - 1} \cdot \frac{1}{m}$$

donde $rank(c_i^t)$ es la posición en la lista ordenada (del mejor al peor individuo) del cromosoma c_i^t , y los coeficientes $1 \leq a_{max} \leq 2$ y $a_{min} = a_{max} - 1$ representan el valor esperado de descendientes del peor y mejor cromosoma de la población respectivamente ($a_{max} = 1, 2$ recomendado).

Otra posibilidad [Michalewicz, 1992] es utilizar un parámetro q introducido por el usuario y definir la probabilidad de selección mediante la siguiente función no lineal:

$$p_i^t = q \cdot (1 - q)^{rank(c_i^t) - 1}$$

donde $0 < q < 1$ es un parámetro definido por el usuario.

Selección por torneo Se selecciona el individuo con mejor adecuación de un grupo de individuos (2 en el torneo binario) elegidos aleatoriamente de la población.

Los distintos métodos de selección pueden ser analizados en términos de su *presión selectiva*. Una medida de ésta es el inverso del tiempo requerido por el mejor individuo para llenar la población con copias de sí mismo, cuando no actúa otro operador genético. Los métodos de selección se ordenan en orden creciente de presión selectiva (para valores estándares de sus parámetros), del siguiente modo: selección proporcional, selección por ranking, selección por torneo.

Muestreo estocástico universal La idea básica de este método (véase el Algoritmo 11.2) es construir una rueda de ruleta con m punteros distribuidos equidistantemente, en lugar de un solo puntero como ocurre con la rueda de ruleta convencional. De esta forma, con un simple “giro” de la ruleta se seleccionan m individuos a la vez.

Sustitución steady-state Como mecanismo de sustitución generacional alternativo a la sustitución generacional completa, podemos destacar la *sustitución de estado estable (steady-state)*. Con esta técnica, se selecciona un número $n \in [1..m]$ de individuos los cuales serán sustituidos por n nuevos. Para la elección de los individuos que mueren suele utilizarse un ranking inverso, es decir, ordenando la lista de cromosomas de menor a mayor adecuación. Este técnica supone una familia de estrategias de sustitución generacional. La sustitución generacional completa pertenece a esta familia con $n = m$. El modelo del factor de crowding es también un caso particular con la particularidad que comentamos a continuación.

Modelo del factor del crowding Un individuo nuevo sustituye a uno viejo el cual es seleccionado de entre un subconjunto de CF miembros elegidos aleatoriamente de la población completa. En este subconjunto se selecciona para morir el individuo más parecido al nuevo, usando como medida un contador de semejanza bit-a-bit.

Algoritmo 11.2 Muestreo estocástico universal.

```
1: sum ← 0;
2: ptr ← rand() ∈ [0, 1];
3: para i = 1 hasta m hacer
4:   sum ← sum + Nit; /* Nit = m · pit */
5:   mientras sum ≥ ptr hacer
6:     seleccionar individuo i;
7:     ptr ← ptr + 1;
8:   fin mientras
9: fin para
```

11.4.3 Operadores de variación

Se han descrito multitud de variantes para los operadores de variación básicos en CE. Aunque muchas variantes se han llevado a cabo en representaciones binarias, es en las representaciones de alta cardinalidad en donde se observa una mayor diversidad de técnicas. Podemos destacar los siguientes operadores:

Operadores de cruce:

- *Cruce uniforme* (rep. binarias y reales). Cada gen en los hijos se crea copiando el correspondiente gen de un padre u otro, utilizando para ello una máscara de cruce generada aleatoriamente. Donde hay un 1 en la máscara, los genes en el primer hijo se toman del primer padre, y donde hay un 0 los genes se toman del segundo padre. Los genes del segundo hijo se establecen con las decisiones inversas.
- *Cruce aritmético* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se generan dos hijos $c_i'^t = (b_{i1}'^t \dots b_{ilong}'^t)$ y $c_j'^t = (b_{j1}'^t \dots b_{jlong}'^t)$, con $b_{il}'^t = a \cdot b_{il}^t + (1-a) \cdot b_{jl}^t$, $b_{jl}'^t = a \cdot b_{jl}^t + (1-a) \cdot b_{il}^t$, para $l = 1, \dots, long$, donde $a \in [0, 1]$ es un número aleatorio.
- *Cruce plano* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se genera el cromosoma $c_k^t = (b_{k1}^t \dots b_{klong}^t)$, con b_{kl}^t generado aleatoria y uniformemente en el intervalo $[b_{il}^t, b_{jl}^t]$, para $l = 1, \dots, long$.
- *Cruce BLX- α* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se genera el cromosoma $c_k^t = (b_{k1}^t \dots b_{klong}^t)$, con b_{kl}^t generado aleatoria y uniformemente en el intervalo $[min_{kl}^t - I\alpha, max_{kl}^t + I\alpha]$, con $min_{kl}^t = min(b_{il}^t, b_{jl}^t)$, $max_{kl}^t = max(b_{il}^t, b_{jl}^t)$, $I = max_{kl}^t - min_{kl}^t$, para $l = 1, \dots, long$. El cruce plano es un caso particular del cruce BLX- α para $\alpha = 0,0$.
- *Cruce PMX, OX y CX*. Usados cuando se utilizan permutaciones como representación (véase la sección 11.7).

Operadores de mutación:

- *Mutación uniforme* (rep. reales). El operador de mutación uniforme mostrado anteriormente para representaciones binarias puede ser definido también para representaciones reales cambiando el gen a mutar por un valor generado aleatoriamente en el dominio de la variable.
- *Mutación no uniforme* (rep. reales). Dado el cromosoma $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$, y supuesto que el gen b_{il}^t ha sido seleccionado para la mutación, el resultado es $c'_i^t = (b_{i1}^t \dots b_{il}^{t'} \dots b_{ilong}^t)$, donde $b_{il}^{t'}$ es, para representaciones reales:

$$b_{il}^{t'} = \begin{cases} b_{il}^t + \Delta(t, UB - b_{il}^t) & \text{si } a = 0 \\ b_{il}^t - \Delta(t, b_{il}^t - LB) & \text{si } a = 1 \end{cases}$$

donde UB y LB son los límites superior e inferior respectivamente del dominio de b_{il}^t , la función $\Delta(t, y)$ devuelve un valor en el rango $[0, y]$ de forma que la probabilidad de que éste sea cercano a cero aumente conforme el algoritmo avanza (en [Michalewicz, 1992] se toma $\Delta(t, y) = y \cdot (1 - r^{(1-\frac{1}{T})^b})$, donde r es un número aleatorio en el intervalo $[0, 1]$, T es el número máximo de generaciones y $b = 5$ determina el grado de dependencia con respecto a la generación actual t), y a es un valor booleano aleatorio.

La mutación no uniforme ha sido definida también para representaciones binarias en [Michalewicz, 1992].

- *Mutación por intercambio*. Se intercambian dos elementos elegidos aleatoriamente de una permutación (véase la sección 11.7).

11.4.4 Manejo de restricciones

El manejo de restricciones no triviales no es fácil de implementar en CE. Básicamente, hay dos enfoques para manejar individuos no factibles, dependiendo de que se permita o no la presencia de individuos no factibles en la población.

Entre los que no permiten individuos no factibles en la población, podemos citar los *métodos abortivos*, que simplemente eliminan los individuos ilegales que se generen, y los *métodos anticonceptivos*, que no permiten la generación de individuos ilegales. Así pues, en determinados problemas, como son el problema del transporte o el problema del viajante de comercio [Michalewicz, 1992], el manejo de restricciones se puede llevar a cabo incorporando el conocimiento específico del problema a la representación, y diseñando métodos de inicialización de soluciones y operadores de variación que exploten tal conocimiento para garantizar que las soluciones generadas satisfacen las restricciones. Los *algoritmos de reparo* pueden usarse para transformar las soluciones no factibles en factibles, y son normalmente complejos y muy dependientes del problema.

En el otro grupo de técnicas, las que permiten la presencia de individuos no factibles en la población, podemos citar como más importantes las *funciones de penalización*. Las técnicas de penalización [Goldberg, 1989a] consisten básicamente en

degradar o penalizar la adecuación de las soluciones en relación al grado de violación de las restricciones. Así, dado el siguiente problema general de optimización con restricciones:

$$\text{Minimizar } f(\vec{x}), \text{ sujeto a } g_i(\vec{x}) \leq 0, i = 1, \dots, n \quad (11.1)$$

donde \vec{x} un vector de m variables, lo podemos transformar al siguiente problema no restringido:

$$\text{Minimizar } f(\vec{x}) + r \cdot \sum_{i=1}^n \Phi[g_i(\vec{x})]$$

donde Φ es la función de penalización y r es el coeficiente de penalización. Para la función de penalización existen muchas alternativas, siendo una de las más usadas $\Phi[g_i(\vec{x})] = g_i^2(\vec{x})$. El coeficiente de penalización r normalmente se fija de forma separada para cada tipo de restricción. Esta técnica trabaja bien excepto para problemas altamente restringidos, en los cuales, encontrar una solución factible es tan difícil como encontrar una solución óptima. Encontrar coeficientes de penalización adecuados para un problema concreto en CE suele ser una tarea difícil y dependiente del problema. Si se usan penalizaciones muy altas, el AE puede perder la mayoría de su tiempo evaluando soluciones ilegales, y además, puede ocurrir que para encontrar una buena solución se tenga que pasar por soluciones ilegales como intermedias, las cuales han sido altamente penalizadas y por tanto eliminadas de la población. Si se usan violaciones moderadas, el sistema puede tener mejor consideración con soluciones ilegales que con otras que no lo son. Esto hace [Michalewicz, 1992] que los AE, considerados como métodos de búsqueda débiles, se conviertan paradógicamente en métodos fuertes (dependientes del problema) en presencia de restricciones no triviales.

A continuación, describimos una regla heurística que se ha mostrado muy eficiente en un gran número de problemas test con restricciones como en la Expresión 11.1, permitiendo la existencia de individuos factibles y no factibles en la población [Jiménez y Verdegay, 1999]:

- Un individuo factible \vec{x} es mejor que un individuo factible \vec{x}' si: $f(\vec{x}) < f(\vec{x}')$
- Un individuo factible es mejor que un individuo no factible.
- Un individuo no factible \vec{x} es mejor que un individuo no factible \vec{x}' si:

$$\max_i\{g_i(\vec{x})\} < \max_i\{g_i(\vec{x}')\}$$

La ventajas de esta heurística radican en que no es dependiente del problema, e integra los conceptos de optimalidad y factibilidad en un mismo proceso evolutivo, en el cual, los individuos no factibles evolucionan hacia la factibilidad, y los individuos factibles evolucionan hacia la optimalidad. Además, la heurística resulta fácil de implantar con algunos esquemas básicos de selección, muestreo o sustitución generacional. Así pues, en un esquema de selección por ranking, los individuos factibles

precederían a los individuos no factibles en el ranking establecido, y dentro de cada grupo los individuos estarían ordenados por las funciones $f(\vec{x})$ y $\max_i\{g_i(\vec{x})\}$ respectivamente. En una selección por torneo, el ganador del grupo se obtiene directamente con la aplicación de la regla heurística. La regla heurística se caracteriza también por ser generalizable en un entorno de optimización multiobjetivo, como veremos en el siguiente apartado.

11.4.5 Optimización multiobjetivo

La mayor parte de los problemas de optimización del mundo real son naturalmente multiobjetivo. Esto es, suelen tener dos o más funciones objetivo que deben satisfacerse simultáneamente y que posiblemente están en conflicto entre sí. Sin embargo, con la finalidad de simplificar su solución, muchos de estos problemas tienden a modelarse como monoobjetivo. De esta forma, con la ayuda de algún conocimiento del problema, los problemas multiobjetivo se transforman usando sólo una de las funciones originales y manejando las adicionales como restricciones, o reduciendo el vector de objetivos a uno solo, el cual optimizar. Con estos enfoques se obtiene una única solución que representa el mejor compromiso entre los objetivos para un ambiente de decisión concreto. Esto en muchas situaciones prácticas es un inconveniente, ya que ante situaciones diferentes del problema se requieren nuevas ejecuciones para la obtención de nuevas soluciones. El hecho de que los AE trabajen con poblaciones de soluciones, va a permitir implementaciones de éstos que permitan la obtención de un conjunto de múltiples soluciones con una sola ejecución del algoritmo, de forma que un decisor pueda elegir, a posteriori, la solución más apropiada de acuerdo al estado de decisión existente, sin necesidad de nuevas ejecuciones.

11.4.5.1 Formulación matemática

Consideraremos la siguiente formulación para un problema de optimización multiobjetivo con restricciones:

$$\begin{aligned} & \text{Minimizar} && f_i(\vec{x}), \quad i = 1, \dots, n \\ & \text{sujeto a:} && g_j(\vec{x}) \leq 0, \quad j = 1, \dots, m \end{aligned} \tag{11.2}$$

donde $\vec{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales $x_k \in \mathbb{R}$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, y $f_i(\vec{x})$, $g_j(\vec{x})$ son funciones arbitrarias. Llamaremos \mathcal{F} al *espacio de las soluciones factibles* o *región factible*, el cual está formado por los vectores \vec{x} que satisfacen las restricciones $g_j(\vec{x}) \leq 0$. \mathcal{S} es el *espacio de búsqueda completa* formado por todos los posibles vectores \vec{x} . El espacio de búsqueda \mathcal{S} se define usualmente como un rectángulo n -dimensional en \mathbb{R}^p . Es claro que $\mathcal{F} \subseteq \mathcal{S}$.

11.4.5.2 Pareto optimalidad

El concepto de Pareto optimalidad fue formulado por Vilfredo Pareto en el siglo XIX y constituye en sí mismo el origen de la optimización multiobjetivo.

Una solución factible $\vec{x} \in \mathcal{F}$ para el problema representado por la Expresión 11.2 *domina* a otra solución $\vec{x}' \in \mathcal{F}$ si:

$$\begin{aligned} f_i(\vec{x}) &\leq f_i(\vec{x}') \quad \text{para todo } i = 1, \dots, n \\ y \quad f_i(\vec{x}) &< f_i(\vec{x}') \quad \text{para al menos un } i = 1, \dots, n \end{aligned}$$

Una solución factible $\vec{x} \in \mathcal{F}$ es Pareto óptima o *no dominada* para el problema planteado si no existe ninguna solución $\vec{x}' \in \mathcal{F}$ tal que \vec{x}' domine a \vec{x} . Dicho de otra forma, una solución es Pareto óptima si no existe ninguna otra solución que mejore en algún objetivo sin empeorar simultáneamente otro objetivo.

El conjunto de soluciones del problema, denominado *conjunto Pareto óptimal* \mathcal{P} está formado por el conjunto de soluciones Pareto óptimas para el problema.

En el espacio de los objetivos, las soluciones no dominadas para el problema planteado son conocidas como el *frente Pareto óptimal* o simplemente *frente Pareto* \mathcal{PF} definido como:

$$\mathcal{PF} = \left\{ \vec{f}(\vec{x}) \mid \vec{x} \in \mathcal{P} \right\}$$

11.4.5.3 Aspectos relevantes en Computación Evolutiva Multiobjetivo

Un aspecto importante en CE Multiobjetivo es la *diversidad*. Las técnicas de diversidad en optimización evolutiva multiobjetivo fueron originalmente sugeridas en [Goldberg, 1989a] a finales de los ochenta y su importancia radica fundamentalmente en dos aspectos. Primero, las múltiples soluciones capturadas mediante el algoritmo deben cubrir el frente Pareto que constituye la solución del problema. Esto significa que el algoritmo debe buscar soluciones no dominadas de una forma diversificada. Segundo, si todos los individuos no dominados de la población tienen la misma probabilidad de selección, es decir, son igual de buenos, esto puede provocar el fenómeno del *genetic drift*, el cual causa que la población converja sólo en una pequeña región del espacio de solución. La técnica de diversidad debe entonces poder discriminar entre dos individuos no dominados, de forma que el individuo no dominado que produzca mejor diversidad tenga mayor probabilidad de selección.

En el diseño de AE multiobjetivo, el concepto Pareto, el manejo de restricciones y la técnica de diversidad deben integrarse junto con el resto de componentes que caracterizan a los AE. La selección por ranking y el torneo binario son los esquemas usados en la literatura, y se ha reconocido ampliamente la importancia del uso de la estrategia elitista. A raíz de la primera propuesta de Goldberg fueron apareciendo multitud de algoritmos multiobjetivo basados en el concepto Pareto que incluían técnicas explícitas de diversidad, dando así lugar a la 1^a generación de algoritmos evolutivos multiobjetivo, donde MOGA, NSGA y NPGA fueron los algoritmos más reconocidos. La inclusión del elitismo en los algoritmos evolutivos multiobjetivo dio lugar a la 2^a generación y última de este tipo de algoritmos; cabe citar a NSGA-II, DPGA, SPEA, SPEA2, PESA, PESA-II, PAES, MOMGA o $M\mu$ GA [Coello y otros, 2002] como los más representativos.

Finalmente, es de extremada importancia para la evaluación de estos algoritmos la disponibilidad de un conjunto de problemas test adecuado. Recientemente, Deb *et al.* [Deb y otros, 2001] proponen un generador de problemas test para optimización

$$\begin{aligned} \text{Minimizar} \quad & f_1(\vec{x}) = x_1 \\ & f_2(\vec{x}) = g(\vec{x}) \left(1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} \right) \\ \text{sujeto a :} \quad & \cos(\theta)(f_2(\vec{x}) - e) - \sin(\theta)f_1(\vec{x}) \geq \\ & a |\sin[b\pi[\sin(\theta)(f_2(\vec{x}) - e) + \cos(\theta)f_1(\vec{x})]^c]|^d \\ & 0 \leq x_1 \leq 1 \\ & -5 \leq x_2, x_3, x_4 \leq 5 \end{aligned}$$

Figura 11.4: Problemas test CTP2, ..., CTP7.

multiobjetivo con restricciones, el cual puede ser configurado para obtener grados de dificultad deseados mediante la elección de diferentes parámetros. La Figura 11.4 muestra un conjunto de estos problemas test, y la Tabla 11.2 los parámetros para la configuración de cada uno de ellos.

Problema	θ	a	b	c	d	e
CTP2	$-0,2\pi$	0,2	10	1	6	1
CTP3	$-0,2\pi$	0,1	10	1	0,5	1
CTP4	$-0,2\pi$	0,75	10	1	0,5	1
CTP5	$-0,2\pi$	0,75	10	2	0,5	1
CTP6	$0,1\pi$	40	0,5	1	2	-2
CTP7	$-0,05\pi$	40	5	1	6	0

Tabla 11.2: Parámetros para los problemas test CTP2, ..., CTP7.

Una revisión completa de CE en optimización multiobjetivo puede encontrarse en [Coello y otros, 2002] y [Deb, 2001]. A continuación describimos el algoritmo NSGA-II, perteneciente a la 2^a generación, el cual presenta actualmente un gran impacto en la comunidad científica.

11.4.5.4 NSGA-II

En [Deb, 2001] se propone NSGA-II, un AG de ordenación no dominada con elitismo. NSGA-II utiliza una estrategia elitista junto con un mecanismo explícito de diversidad. En realidad, este algoritmo se diferencia en muchos aspectos del original NSGA, aunque los autores han decidido mantener el nombre NSGA-II para referenciar su origen.

En NSGA-II, a partir de una población P_t se crea una nueva población de descendientes Q_t aplicando *selección por torneo binario por frentes y nicho*, y los operadores evolutivos de cruce y mutación. Estas dos poblaciones se mezclan para formar una nueva población R_t de tamaño $2N$ (siendo N el tamaño de la población original P_t). La nueva población P_{t+1} se obtiene a partir de la población R_t , de la cual sobreviven los N mejores individuos tras aplicar la (*ordenación por frentes y nicho*).

Dada una población inicial aleatoria P_0 , NSGA-II realiza los siguientes pasos:

1. Crear una nueva población Q_t de N individuos a partir de la población P_t (de N individuos) aplicando *selección por torneo binario por frentes y nicho*, y los operadores evolutivos de cruce y mutación.
2. Combinar las poblaciones P_t y Q_t para crear una nueva población $R_t = P_t \cup Q_t$.
3. Crear una nueva población P_{t+1} seleccionando N individuos de la población R_t utilizando la *ordenación por frentes y nichos*.

La selección por torneo binario por frentes y nicho compara dos soluciones y devuelve aquella mejor según el *operador de no dominancia y nicho*. De igual forma, la ordenación por frentes y nicho ordena los individuos según las condiciones establecidas por el *operador de no dominancia y nicho*.

Operador de no dominancia y nicho Cada individuo i tiene dos atributos:

1. Un rango de no dominancia r_i dentro de la población.
2. Una distancia de nicho local d_i , que es una medida del espacio de búsqueda alrededor de la solución i que no está ocupado por ninguna otra solución de la población.

Un individuo i es preferible a un individuo j si ocurre alguna de las siguientes condiciones:

1. Si el individuo i tiene mejor rango que j , es decir, si $r_i < r_j$.
2. Si los individuos i y j tienen el mismo rango, pero el individuo i tiene mejor *distancia de nicho* que el individuo j , es decir, $r_i = r_j$ y $d_i > d_j$.

La primera condición asegura que la solución escogida se encuentra en un frente no dominado mejor. La segunda condición resuelve el conflicto en caso de que ambas soluciones se encuentren en el mismo frente, decidiéndose por aquella solución que tenga una mejor distancia de nicho, asegurando por tanto la diversidad.

Distancia de nicho La distancia de nicho d_i se calcula en NSGA-II de la siguiente forma:

- Sea \mathcal{F}_i el conjunto de individuos de la población que pertenecen al mismo frente Pareto que el individuo i .
- Para cada función objetivo $j = 1, \dots, m$:
 - Ordenar los individuos de \mathcal{F}_i según el valor de la función objetivo f_j en una lista l_j^i .
 - f_j^{i+1} es el valor de la función j para el individuo en la lista l_j^i siguiente al individuo i .

- f_j^{i-1} es el valor de la función j para el individuo en la lista l_j^i anterior al individuo i .
- Calcular el valor de la distancia d_i de la forma:

$$d_i = \sum_{j=1}^m \frac{f_j^{i+1} - f_j^{i-1}}{f_j^{\max} - f_j^{\min}}$$

donde f_j^{\max} y f_j^{\min} son los valores máximos y mínimos para la función j en toda la población.

Esta métrica denota el valor normalizado de la mitad del perímetro del hipercubo definido por las soluciones más cercanas situadas como vértices.

La Figura 11.5 muestra los resultados obtenidos con NSGA-II para el problema test CTP2.

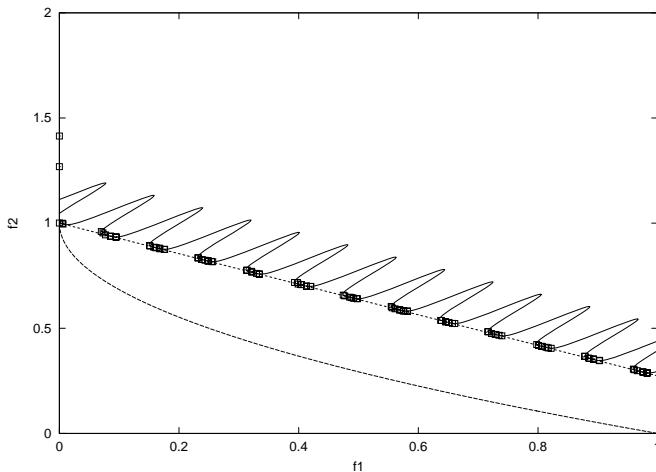


Figura 11.5: Soluciones no dominadas obtenidas con NSGA-II para el problema test CTP2.

11.4.5.5 Métricas de evaluación

Para evaluar los algoritmos evolutivos multiobjetivo, en [Zitzler y otros, 2003] se propone el uso de la métrica ν que calcula la fracción del espacio que no es dominado por ninguna de las soluciones obtenidas por los algoritmos. El objetivo es minimizar el valor de ν . Esta métrica estima tanto la distancia de las soluciones al frente Pareto real como la diversidad de las mismas. El valor de ν puede calcularse como se muestra en la Expresión 11.3, donde P' está compuesta por las N' soluciones no dominadas de P , y f_j^{\max} , f_j^{\min} son los valores máximos y mínimos utópicos para el objetivo j -ésimo respectivamente.

$$\nu = 1 - \frac{\sum_{i=1}^{N'} \left[(f_n^{umax} - f_n^i) \prod_{j=1}^{n-1} \left(f_j^{sup_j^i} - f_j^i \right) \right]}{\prod_{j=1}^n (f_j^{umax} - f_j^{umin})} \quad (11.3)$$

Con el objetivo de realizar una evaluación más exacta de los algoritmos evolutivos multiobjetivo, se han propuesto muchas otras métricas, tanto para evaluar la convergencia como la diversidad. En [Deb, 2001] se recoge un amplio conjunto de dichas métricas. En esta sección mostramos dos de ellas.

La primera métrica, la distancia generacional Υ , evalúa la proximidad de la población al frente Pareto óptimo calculando la distancia media de la población a una población ideal P^* compuesta por N^* soluciones distribuidas uniformemente en el frente Pareto. Esta métrica se muestra en la Expresión 11.4.

$$\Upsilon = \frac{\left(\sum_{i=1}^{N'} d_i^v \right)^{1/v}}{N'} \quad (11.4)$$

En el caso más simple, $v = 1$. El parámetro d_i es la distancia Euclídea (en el espacio objetivo) entre la solución i y la solución más próxima en P^* :

$$d_i = \min_{k=1}^{N^*} \sqrt{\sum_{j=1}^n (f_j^i - f_j^{*k})^2}$$

donde f_j^{*k} es el valor del objetivo j -ésimo de la solución k -ésima en P^* .

La segunda métrica utilizada es la dispersión Δ para evaluar la diversidad de la población. La Expresión 11.5 muestra esta métrica.

$$\Delta = \frac{\sum_{j=1}^n d_j^e + \sum_{i=1}^N |d_i - \bar{d}|}{\sum_{j=1}^n d_j^e + N\bar{d}} \quad (11.5)$$

donde d_i puede ser cualquier medida de la distancia entre soluciones adyacentes, por ejemplo, la distancia Euclídea, y \bar{d} es el valor medio de dicha medida. El parámetro d_j^e es la distancia entre las soluciones extremas en P^* y P correspondientes al objetivo j -ésimo.

11.4.6 Evaluación y validación de algoritmos evolutivos

A la hora de evaluar y validar un AE, hay que tener en cuenta las siguientes consideraciones generales:

- Obtener los valores mejor, medio, peor y varianza de sobre al menos 100 ejecuciones del algoritmo.
- Para la comparación con otras técnicas evolutivas, se considerarán los resultados para un mismo número de evaluaciones de la función objetivo, e iguales valores de los parámetros (tamaño de la población y probabilidades de aplicación de los operadores de variación).
- Usar un conjunto suficientemente amplio de problemas test, variando, si fuese posible, el grado de dificultad, así como el tamaño del problema (escalabilidad).
- Probar esquemas alternativos de selección, muestreo, sustitución generacional y operadores de variación.
- Realizar un proceso previo de experimentación para la elección de los parámetros. Para ello, se requerirán múltiples ejecuciones del algoritmo para cubrir las distintas combinaciones de los valores de los parámetros, fijando finalmente éstos a aquellos que hayan producido mejores resultados en el proceso de experimentación.
- Calcular el número de violaciones y su cuantía en optimización con restricciones.
- Usar métricas de diversidad y optimalidad en optimización multiobjetivo.
- Realizar análisis estadísticos de los resultados.
- Comparar los resultados con otras técnicas existentes, tanto deterministas como probabilistas.

11.5 Lecturas recomendadas

El libro [Goldberg, 1989a] sigue siendo una referencia en investigaciones recientes. Es un libro que tiene un carácter tanto divulgativo como de iniciación a la investigación, si bien puede resultar en determinados aspectos algo difícil de leer y asimilar. El lector debe ser consciente de que algunos de los conceptos que ahí aparecen han sido objeto de numerosos e intensivos estudios en los años posteriores, por lo que deberán ser tenidos en cuenta sólo con carácter introductorio.

Un libro más ameno de leer y estudiar es [Michalewicz, 1992], el cual tiene también un carácter tanto docente como investigador. Este libro se centra más en las aplicaciones y en el diseño de algoritmos mediante el uso de representaciones alternativas específicas del problema. No es un libro tan técnico como el anterior, y resulta muy apropiado como revisión de los trabajos realizados hasta el momento de su publicación. Existen ediciones posteriores a 1992 más actualizadas.

Los libros [Coello y otros, 2002] y [Deb, 2001] son excelentes libros de introducción a la optimización evolutiva multiobjetivo y al manejo de restricciones, representando asimismo una referencia obligada para los investigadores en el campo. No sólo se introducen los conceptos fundamentales que giran alrededor de la optimización evolutiva

multiobjetivo con restricciones, sino que también se repasan las técnicas evolutivas que más impacto han tenido en el campo por su eficiencia y aplicabilidad, incluyendo un exhaustivo análisis comparativo. El prestigio de los autores avala la calidad de estos libros.

11.6 Resumen

El capítulo hace un repaso a las principales técnicas que intervienen en el diseño de algoritmos evolutivos. Tras una breve introducción histórica a la Computación Evolutiva, y una clasificación y caracterización, se describe el algoritmo genético simple que D. Goldberg implementó en sus primeros trabajos, y que supuso el comienzo de esta importante disciplina. Las técnicas más usadas en la literatura para representación, selección, muestreo y sustitución generacional, así como los operadores de variación, son descritas de forma sencilla pensando en su utilización por parte del lector en el diseño de nuevos algoritmos evolutivos. Aspectos importantes que surgen en optimización, tales como el manejo de restricciones y la optimización multiobjetivo, son tratados también desde el punto de vista de su integración en el diseño de algoritmos evolutivos. Dos ejemplos de algoritmos evolutivos que usan representaciones *ad hoc* son descritos, uno para el conocido problema del viajante de comercio, y otro para el problema de la identificación de variables que surge en Minería de Datos. Finalmente, se propone un interesante problema de búsqueda, el Sudoku, que permitirá al lector comprobar su capacidad en el diseño de algoritmos evolutivos mediante el uso de las técnicas aprendidas.

11.7 Ejercicios resueltos

11.1. El Problema del Viajante de Comercio. El *problema del viajante de comercio*, también conocido por sus siglas en inglés como TSP, es uno de los problemas más famosos y mejor estudiados en el campo de la optimización combinatoria computacional. A pesar de la aparente sencillez de su planteamiento, el TSP es uno de los más complejos de resolver y existen demostraciones que equiparan la complejidad de su solución a la de otros problemas aparentemente mucho más complejos. El TSP está entre los problemas denominados *NP-completos*, esto es, los problemas que no se pueden resolver en tiempo polinomial en función del tamaño de la entrada. Sin embargo, algunos casos concretos del problema sí han sido resueltos hasta su optimización, lo que le convierte en un excelente banco de pruebas para algoritmos de optimización.

Dadas n ciudades de un territorio, y la distancia entre cada ciudad, el problema consiste en encontrar una ruta que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajante. Es decir, encontrar una permutación $P = \{c_0, c_1, \dots, c_{n-1}\}$ que minimice la función:

$$d_P = \sum_{i=0}^{n-1} d(c_i, c_{(i+1) \bmod n})$$

donde $d(x, y)$ es la distancia entre la ciudad x y la ciudad y .

Una formulación equivalente en términos de la teoría de grafos es la de encontrar, en un grafo completamente conexo y con arcos ponderados, el ciclo hamiltoniano de menor coste. En esta formulación cada vértice del grafo representa una ciudad, cada arco representa una carretera y el peso asociado a cada arco representa la longitud de la carretera.

Dada la explosión combinatoria de las posibles soluciones, los algoritmos clásicos no son capaces de resolver el problema general. Por ello, a su solución se han aplicado distintas técnicas computacionales, normalmente basadas en heurísticas, tales como los AE. Sin embargo, desde el punto de vista teórico, estas aproximaciones no suponen una resolución real del TSP, pero ofrecen soluciones aproximadas suficientemente aceptables.

Quizás la forma más natural de representar una solución en un AE para el TSP [Grefenstette, 1987] sea la representación mediante una ruta. De esta forma, un individuo:

$$(3 \ 1 \ 4 \ 6 \ 9 \ 8 \ 5 \ 2 \ 7)$$

representa la solución que, partiendo de una ciudad origen, realiza el camino $3 - 1 - 4 - 6 - 9 - 8 - 5 - 2 - 7$, para retornar finalmente a la ciudad de origen.

La representación de las soluciones es por tanto una permutación de números enteros que representan ciudades, en la cual cada entero aparece una sola vez. Es fácil diseñar un algoritmo de inicialización aleatoria de soluciones mediante este tipo de representación.

Establecida la representación de soluciones y su inicialización, el siguiente paso es diseñar operadores de variación que se adapten a la representación. Se han descrito multitud de operadores de cruce, mutación y otros, siendo los más usados el cruce PMX, la mutación por intercambio recíproco, y la inversión.

Con el *cruce PMX*, dados dos padres seleccionados para la operación de cruce, se establece una subsecuencia mediante la elección de dos puntos de cruce, y los hijos resultantes contienen las subsecuencias intercambiadas de los padres, y el resto de elementos son cambiados, mediante un algoritmo de reparo, de forma que se preserve, por un lado, el máximo de información (ciudad y orden) de los padres, y por otro, las restricciones impuestas por la representación (cada ciudad debe aparecer una sola vez).

Dados dos padres y los puntos de cruce:

$$p_1 = (1 \ 2 \ 3 \mid 4 \ 5 \ 6 \ 7 \mid 8 \ 9)$$

$$p_2 = (4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3)$$

se producen dos hijos, en principio ilegales, de la siguiente forma:

$$\begin{aligned} h_1 &= (1 \ 2 \ 3 \mid 1 \ 8 \ 7 \ 6 \mid 8 \ 9) \\ h_2 &= (4 \ 5 \ 2 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3) \end{aligned}$$

El algoritmo de reparo comienza estableciendo los siguientes mapeos:

$$1 \leftrightarrow 4, \ 8 \leftrightarrow 5, \ 7 \leftrightarrow 6, \ 6 \leftrightarrow 7$$

Las ciudades que presentan un conflicto son entonces cambiadas siguiendo la tabla de mapeos. Así, las ciudades 1 y 8 en h_1 son cambiadas por las ciudades 4 y 5 respectivamente de acuerdo al mapeo establecido. De igual forma, las ciudades 4 y 5 en h_2 son cambiadas por la 1 y la 8 respectivamente. Los descendientes resultantes son:

$$\begin{aligned} h_1' &= (4 \ 2 \ 3 \mid 1 \ 8 \ 7 \ 6 \mid 5 \ 9) \\ h_2' &= (1 \ 8 \ 2 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3) \end{aligned}$$

los cuales explotan importantes similaridades en el valor y orden simultáneamente de los padres. Con la *mutación por intercambio recíproco*, se eligen dos ciudades aleatoriamente, y se intercambian, asegurando así una solución legal:

$$\begin{aligned} h_1 &= (4 \ 2 \ 3 \ 1 \ 8 \ 7 \ 6 \ 5 \ 9) \\ &\quad \downarrow \\ h_1' &= (4 \ 9 \ 3 \ 1 \ 8 \ 7 \ 6 \ 5 \ 2) \end{aligned}$$

Otro operador unario usado bajo esta representación es la *inversión*, si bien su eficacia en el proceso evolutivo ha sido más cuestionada. La inversión simple establece dos puntos aleatorios entre los cuales el orden de las ciudades es invertido, garantizando una solución válida:

$$\begin{aligned} h_1' &= (4 \ 9 \mid 3 \ 1 \ 8 \ 7 \ 6 \mid 5 \ 2) \\ &\quad \downarrow \\ h_1'' &= (4 \ 9 \mid 6 \ 7 \ 8 \ 1 \ 3 \mid 5 \ 2) \end{aligned}$$

■

11.2. Selección de variables en Minería de Datos. La propuesta de Zadeh de modelar el mecanismo del pensamiento humano con valores lingüísticos difusos en vez de con números, condujo a la introducción de la borrosidad dentro de la teoría de sistemas y al desarrollo de una nueva clase de sistemas llamados sistemas difusos. En el modelado difuso, el problema más importante es la identificación de un modelo

difuso usando datos del tipo entrada-salida. De un modo similar, la *Minería de Datos* (MD) tiene como objetivo la criba de datos para revelar información útil por medio de una representación adecuada al usuario, comprimiendo enormes registros de datos. En este contexto, un problema común a ambas técnicas es cómo tratar con mecanismos de selección de variables a partir de la posible colección de variables que pueden ser consideradas de los datos disponibles.

Aunque hemos usado el término MD, utilizamos el término *Descubrimiento de Conocimiento en Bases de Datos* (DCBD) o sólo *Descubrimiento de Conocimiento* (DC) para denotar el proceso completo de extraer conocimiento de alto nivel a partir de datos de bajo nivel, y el término MD como el acto concreto de extraer patrones o modelos de los datos. Por tanto, muchos pasos preceden al de MD, y uno de ellos es el preprocesamiento de los datos para seleccionar las variables adecuadas para ser usadas en la construcción o extracción del modelo. Este paso es similar a la identificación de variables dentro del paso de identificación de la estructura, en el proceso de *Modelización Difusa*.

Dado un conjunto de datos para el que presumimos alguna dependencia funcional, surge la cuestión de si hay alguna metodología adecuada para derivar reglas (difusas) a partir de los datos que caracterizan la función desconocida, de forma tan precisa como sea posible. Recientemente se han propuesto numerosas aproximaciones para generar automáticamente reglas if-then a partir de datos numéricos sin expertos en el dominio. Este tipo de problema es similar al que podemos encontrar en el área de DC, y en este sentido describimos aquí una técnica *Soft Computing* que intenta tener en cuenta uno de los problemas fundamentales como es el preprocesamiento de los datos para seleccionar las características o variables adecuadas para que el proceso de MD pueda realizarse con información más relevante.

A medida que intentamos resolver problemas del mundo real, nos damos cuenta de que son normalmente sistemas mal definidos, difíciles de modelar y con espacios de solución de gran escala. En estos casos, los modelos precisos son poco prácticos, demasiado caros o inexistentes. La información relevante disponible está normalmente en la forma de conocimiento empírico previo y datos del tipo entrada-salida representando instancias del comportamiento del sistema. Así pues, necesitamos sistemas de razonamiento aproximado capaces de manejar esa información imperfecta. Soft Computing es un término acuñado recientemente que describe el uso simbiótico de muchas disciplinas emergentes de computación que intentan manejar la información imperfecta. De acuerdo con Zadeh (1994): "... en contraste con lo tradicional, hard computing, soft computing es tolerante a la imprecisión, a la incertidumbre y a la verdad parcial". Dentro de estas técnicas tenemos la Lógica Difusa, el Razonamiento Probabilístico, las Redes Neuronales y los Algoritmos Evolutivos. Durante los últimos años hemos visto un número creciente de algoritmos híbridos, en los que dos o más tecnologías Soft Computing se han integrado para mejorar el rendimiento global del algoritmo.

Como sucede en la MD, si intentamos revelar información útil de los datos, un paso fundamental es el preprocesamiento de los datos para seleccionar las variables más adecuadas. En el contexto del modelado y más concretamente de la modelización

difusa, el objetivo es encontrar un conjunto de relaciones que describan el comportamiento presente en los datos por medio de una colección de patrones o reglas if-then. En este proceso, la identificación de la estructura de un sistema tiene que encontrar las variables que representan los datos de un modo más preciso, de entre una colección de posibles variables. En este contexto tenemos que seleccionar un número finito de variables entre una colección finita de posibles candidatos. Esto es un problema combinatorio. Una posible aproximación a este problema es intentar asignar cierto grado a cada variable dependiendo de su importancia en la consecución del objetivo final. Este proceso es similar a una fusión multisensor, que consiste en una combinación de diferentes fuentes de información en un formato de representación. Cuando cada variable representa características diferentes, tratamos con información complementaria. Su fusión nos permite resolver ambigüedad en la información.

En este proceso de fusión de las diferentes variables de entrada posibles, tenemos dos finalidades. Primero queremos determinar la importancia de las entradas a fusionar; segundo, queremos determinar los parámetros exactos de las funciones de agregación usadas en la fusión. Encontrar los mejores parámetros de la función de agregación es un proceso de optimización. Los AE han demostrado ser muy útiles para este proceso porque aportan una búsqueda robusta en espacios de búsqueda complejos y no quedan atrapados en mínimos locales como les sucede a las técnicas de gradiente descendente.

Una vez que las variables son identificadas, el siguiente paso en la identificación de la estructura tiene que ver con la identificación de relaciones o, en otras palabras, con el descubrimiento de los patrones o el modelo a partir de los datos, y su representación por medio de diferentes alternativas como por ejemplo reglas difusas. En este paso se pueden usar diferentes técnicas Soft Computing en el contexto del DC.

En este apartado describimos un AE para identificación de variables en Minería de Datos y DC. Expresado en términos de programación matemática, el problema puede ser formulado como sigue:

$$\begin{aligned} & \text{Minimizar} \quad E = \sqrt{\frac{\sum_{t=1}^n \left(y_t - \left(\sum_{j=1}^p w_j x_j^{tf} \right)^{1/f} \right)^2}{n}} \\ & \text{sujeto a :} \quad \sum_{j=1}^p w_j = 1, \quad 0 \leq w_j \leq 1, \quad j = 1, \dots, p \end{aligned} \tag{11.6}$$

donde n es el número de datos, p es el número de variables, y_t es el valor esperado de salida para el vector de variables de entrada $X_t = \{x_1^t, \dots, x_p^t\}$, y $\left(\sum_{j=1}^p w_j x_j^{tf} \right)^{1/f}$ es el valor del modelo *media generalizada* con grado de borrosidad f . Este tipo de operador de agregación es similar al operador OWA (*Ordered Weighted Averaging*) introducido por Yager (1988). Los pesos w_j , $j = 1, \dots, p$, han de ser averiguados.

Aunque pueden usarse numerosas conectivas de conjuntos difusos con la finalidad de agregación, el operador media generalizada satisface las propiedades deseables. Por lo tanto, el operador puede usarse como unión o como intersección en los casos extremos, y el ratio de compensación puede controlarse variando el grado de borrosidad f .

A continuación se describen las principales características del AE propuesto. Estas características son una representación de soluciones al problema, satisfacción de restricciones, mecanismos para crear una población inicial de soluciones, la función de evaluación, operadores de variación y parámetros usados.

Representación Un individuo W de la población se representa como una colección de p elementos $W = \{w_1, \dots, w_p\}$, donde $w_j \in [0, 1]$, $j = 1, \dots, p$, representa el peso asociado a la variable de entrada x_j .

Satisfacción de restricciones Todos los mecanismos para crear un nuevo individuo $W = \{w_1, \dots, w_p\}$ en el proceso evolutivo, es decir los procedimientos de inicialización y los operadores de variación, aseguran que se satisfacen la restricción

$$\sum_{j=1}^p w_j = 1, \quad 0 \leq w_j \leq 1, \quad j = 1, \dots, p.$$

Algoritmo 11.3 Población Inicial.

Entrada: tamaño población $tampob$; número de variables p ;
Salida: población $POB = \{W_1 = \{w_1^1, \dots, w_p^1\}, \dots, W_{tampob} = \{w_1^{tampob}, \dots, w_p^{tampob}\}\}$ de $tampob$ individuos tales que $w_j^i \in [0, 1]$, $j = 1, \dots, p$, $i = 1, \dots, tampob$, y $\sum_{j=1}^p w_j^i = 1$, $i = 1, \dots, tampob$;

```

1: para  $i = 1$  hasta  $tampob$  hacer
2:    $aleat \leftarrow$  valor real aleatorio  $\in [0, 1]$ ;
3:   si  $aleat \leq 0,5$ ; entonces
4:      $pesos1(entrada : p, 1; salida : W)$ ;
5:   si no
6:      $pesos2(entrada : p, 1; salida : W)$ ;
7:   fin si
8:    $W_i \leftarrow W$ ;
9: fin para
```

Población inicial El Algoritmo 11.3 obtiene una población completa de $tampob$ individuos que satisfacen las restricciones impuestas. Consideramos dos procedimientos de inicialización. El procedimiento $pesos1$ (véase el Algoritmo 11.4) genera una

colección de pesos $W = \{w_1, \dots, w_q\}$ tales que $w_l \in [0, 1]$, $l = 1, \dots, q$, y $\sum_{l=1}^q w_l = val$,

con $0 \leq val \leq 1$. El procedimiento $pesos2$ (véase el Algoritmo 11.5) es una modificación del procedimiento $pesos1$ para fijar una proporción aleatoria de pesos iguales

a cero. Nótese que para $q = p$ y $val = 1$, ambos procedimientos pesos1 y pesos2 generan una solución factible para el problema. Estos procedimientos se usan con igual probabilidad para obtener la población inicial.

Algoritmo 11.4 pesos1.

Entrada: número entero q , con $1 \leq q \leq p$; valor real val , con $0 \leq val \leq 1$;

Salida: colección $W = \{w_1, \dots, w_q\}$ tal que $w_l \in [0, 1]$, $l = 1, \dots, q$, y $\sum_{l=1}^q w_l = val$;

- 1: $w_l \leftarrow$ valor real aleatorio $\in [0, 1]$, para $l = 1, \dots, q$;
 - 2: $w_l \leftarrow val \cdot w_l / \sum_{i=1}^q w_i$, para $l = 1, \dots, q$;
-

Algoritmo 11.5 pesos2.

Entrada: número entero q , con $1 \leq q \leq p$; valor real val , con $0 \leq val \leq 1$;

Salida: colección $W = \{w_1, \dots, w_q\}$ tal que $w_l \in [0, 1]$, $l = 1, \dots, q$, y $\sum_{l=1}^q w_l = val$;

- 1: Seleccionar aleatoriamente $K = \{k_1, \dots, k_r\} \subseteq \{1, \dots, q\}$ tal que $1 \leq r \leq q$;
 - 2: $M \leftarrow \{1, \dots, q\} - K$;
 - 3: $w_l \leftarrow 0$, $l \in M$;
 - 4: pesos1(entrada : $r, val; salida : V$);
 - 5: $w_{k_l} \leftarrow v_l$, $l = 1, \dots, r$;
-

Función de evaluación La función de evaluación de los individuos $W_i = \{w_1^i, \dots, w_p^i\}$, $i = 1, \dots, tampob$, está determinada por la función objetivo del problema:

$$eval(W_i) = \sqrt{\frac{\sum_{t=1}^n \left(y_t - \left(\sum_{j=1}^p w_j^i x_j^{tf} \right)^{1/f} \right)^2}{n}}, \quad i = 1, \dots, tampob$$

Operadores de variación Hemos considerado tres operadores de mutación y un operador de cruce. El operador *mutación1*, Algoritmo 11.6, realiza un cambio mínimo, intercambiando dos elementos aleatorios de los padres, mientras que los operadores *mutación2*, Algoritmo 11.7, y *mutación3*, Algoritmo 11.8, usan los procedimientos *pesos1* y *pesos2* respectivamente para producir un cambio en una parte arbitraria de los padres. El operador *cruce_aritmético*, Algoritmo 11.9), produce dos descendientes mediante la combinación lineal convexa de los padres.

Algoritmo 11.6 mutación1.

Entrada: $\text{padre } W = \{w_1, \dots, w_p\}$;
Salida: $\text{descendiente } W' = \{w'_1, \dots, w'_p\}$;

- 1: Seleccionar aleatoriamente $K = \{k_1, k_2\} \subseteq \{1, \dots, p\}$;
- 2: $M \leftarrow \{1, \dots, p\} - K$;
- 3: $w'_l \leftarrow w_l, l \in M$;
- 4: $w'_{k_1} \leftarrow w_{k_2}$;
- 5: $w'_{k_2} \leftarrow w_{k_1}$;

Algoritmo 11.7 mutación2.

Entrada: $\text{padre } W = \{w_1, \dots, w_p\}$;
Salida: $\text{descendiente } W' = \{w'_1, \dots, w'_p\}$;

- 1: Seleccionar aleatoriamente $K = \{k_1, \dots, k_r\} \subseteq \{1, \dots, p\}$ tal que $1 < r \leq p$;
- 2: $M \leftarrow \{1, \dots, p\} - K$;
- 3: $w'_l \leftarrow w_l, l \in M$;
- 4: $val \leftarrow \sum_{l=1}^r w_{k_l}$;
- 5: $\text{pesos1}(\text{entrada} : r, val; \text{salida} : V)$;
- 6: $w'_{k_l} \leftarrow v_l, l = 1, \dots, r$;

Algoritmo 11.8 mutación3.

Entrada: $\text{padre } W = \{w_1, \dots, w_p\}$;
Salida: $\text{descendiente } W' = \{w'_1, \dots, w'_p\}$;

- 1: Seleccionar aleatoriamente $K = \{k_1, \dots, k_r\} \subseteq \{1, \dots, p\}$ tal que $1 < r \leq p$;
- 2: Set $M = \{1, \dots, p\} - K$;
- 3: $w'_l \leftarrow w_l, l \in M$;
- 4: $val \leftarrow \sum_{l=1}^r w_{k_l}$;
- 5: $\text{pesos2}(\text{entrada} : r, val; \text{salida} : V)$;
- 6: $w'_{k_l} \leftarrow v_l, l = 1, \dots, r$;

Algoritmo 11.9 cruce aritmético.

Entrada: padres $W_1 = \{w_1^1, \dots, w_p^1\}$ y $W_2 = \{w_1^2, \dots, w_p^2\}$;
Salida: descendencia $W'_1 = \{w'_1^1, \dots, w_p^1\}$ y $W'_2 = \{w'_1^2, \dots, w_p^2\}$;

- 1: $c_1 \leftarrow$ valor real aleatorio $\in [0, 1]$;
- 2: $c_2 \leftarrow 1 - c_1$;
- 3: $w'_j^1 \leftarrow c_1 \cdot w_j^1 + c_2 \cdot w_j^2, j = 1, \dots, p$;
- 4: $w'_j^2 \leftarrow c_2 \cdot w_j^1 + c_1 \cdot w_j^2, j = 1, \dots, p$;

Evaluación Para evaluar el AE hemos considerado un test estándar propuesto en [Dyckhoff y Pedrycz, 1984]. El conjunto de datos consiste en una colección de 2 variables de entrada x_1 y x_2 y una variable de salida. Para comprobar la eficacia del AE, hemos añadido a los datos de entrada 6 nuevas variables x_3, x_4, x_5, x_6, x_7 y x_8 formadas a partir las variables x_1 y x_2 añadiéndoles ruido uniforme en un 10 % (a las variables x_3 y x_4), en un 40 % (a las variables x_5 y x_6), y en un 100 % (a las variables

x_7 y x_8). La Tabla 11.3 muestra los pesos y fitness obtenidos (valores medios sobre 10 ejecuciones) considerando solamente las 2 variables de entrada del problema, para 3 valores diferentes de f (con el objetivo de poder comparar con los ejemplos de la literatura). La Tabla 11.4 muestra los pesos y fitness obtenidos teniendo en cuenta las 2 variables de entrada originales más las 6 variables ficticias de ruido. Los resultados muestran cómo el AE propuesto detecta las variables que realmente influyen en la variable de salida, excluyendo aquellas ficticias introducidas artificialmente.

	$f = 0,39$	$f = 1,25$	$f = 2$
w_1	0.447	0.405	0.316
w_2	0.553	0.595	0.684
fitness	0.053	0.114	0.161

Tabla 11.3: Resultados considerando únicamente la muestra.

	$f = 0,39$	$f = 1,25$	$f = 2$
w_1	0.420	0.406	0.318
w_2	0.486	0.518	0.682
w_3	0.024	0.0	0.0
w_4	0.015	0.076	0.0
w_5	0.002	0.0	0.0
w_6	0.012	0.0	0.0
w_7	0.019	0.0	0.0
w_8	0.022	0.0	0.0
fitness	0.052	0.113	0.161

Tabla 11.4: Resultados considerando la muestra con ruido.

11.8 Ejercicios propuestos

11.1. Sudoku. El Sudoku es el pasatiempo de moda en todo el mundo: se trata de un rompecabezas matemático en el que hay que rellenar los huecos que faltan. El objetivo es llenar un cuadrado de 9×9 celdas (véase la Figura 11.6), dividido a su vez en subcuadrículas de 3×3 , con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunos espacios. Está prohibido repetir un número en una misma fila, columna o subcuadrícula. ¡La solución es única, así que ármate de paciencia y mucha suerte!

1. Proponer un AE para el Sudoku, estableciendo:

- Una representación de soluciones.
- Una función de evaluación.
- Un operador de cruce.
- Un operador de mutación.

5	3		4	8				
		6	5					
8			2	1		5		
	1					9	8	
		2	1	5	9	4		
	5	3					1	
	6		7	4	3			1
				1				
			3	5		7	4	

Figura 11.6: Un sudoku.

2. Implementar distintos AE mediante:

- Selección proporcional, muestreo estocástico con reemplazamiento (rueda de ruleta), elitismo.
- Selección proporcional, muestreo universal estocástico, elitismo.
- Selección por ranking lineal, muestreo universal estocástico, elitismo.
- Selección por torneo binario, elitismo.

3. Evaluar y comparar los distintos AE. Mostrar los valores del mejor, medio, peor y varianza sobre 100 ejecuciones.

Referencias

- COELLO, C.A.; VELDHUIZEN, D.V. y LAMONT, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic/Plenum publishers, New York, 2002.
- DAVIS, L.: *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
- DEB, K.: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, LTD, 2001.
- DEB, K.; PRATAP, A. y MEYARIVAN, T.: «Constrained test problems for multi-objective evolutionary optimization». *Lectures Notes in Computer Science*, 2001, **1993**, pp. 284–298.
- DEJONG, K.A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doctoral Dissertation, University of Michigan, 1975.
- DYCKHOFF, H. y PEDRYCZ, W.: «Generalized means of model of compensative connectives». *Fuzzy Sets and Systems*, 1984, **14**, pp. 143–154.
- FOGEL, L.J.; OWENS, A.J. y WALSH, M.J.: *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.
- GOLDBERG, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989a.
- GOLDBERG, D.E.: «Sizing populations for serial and parallel genetic algorithms». En: California Morgan Kaufmann Inc. J.D. Schaffer (ed.), San Mateo (Ed.), *Proc. of the Third Intern. Conf. on Genetic Algorithms*, pp. 70–79, 1989b.
- GREFENSTETTE, J.J.: «Incorporating problem specific knowledge into genetic algorithms». En: L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Pitman, London, , 1987.
- HOLLAND, J.H.: *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
- JIMÉNEZ, F. y VERDEGAY, J.L.: «Evolutionary techniques for constrained optimization problems». En: *Proc. of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, Springer-Verlag, , 1999.
- KOZA, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1972.
- MICHALEWICZ, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

- RECHENBERG, I.: *Evolutionary Estrategy: Optimization of Technical Systems According to the Priciple of Biological Evolution.* Frommann-Holzboog, 1973.
- ZITZLER, E.; THIELE, L.; LAUMANNS, M.; FONSECA, C.M. y DA FONSECA, V. GRUNERT: «Performance Assessment of Multiobjective Optimizers: An Analysis and Review». *IEEE Transactions on Evolutionary Computation*, 2003, **7(2)**, pp. 117–132.

Parte IV

TAREAS

Capítulo 12

Diagnosis

Carlos Alonso González
Universidad de Valladolid

12.1 Introducción

La diagnosis es una tarea cuyo objetivo es mantener un sistema en estado operativo el mayor tiempo posible. Aunque tradicionalmente ha sido realizada por equipos humanos especializados, el carácter tecnológico de la sociedad actual hace que la necesidad de disponer de sistemas de ayuda al diagnóstico, o incluso de diagnóstico automático, no deje de crecer. En el campo de la ingeniería, ya es común que muchos dispositivos incluyan componentes de detección y diagnóstico. Se pueden encontrar en artefactos sofisticados, como los satélites espaciales; o en productos cotidianos, como las fotocopiadoras o los automóviles. Añaden fiabilidad a estos sistemas y facilitan la reparación al personal de mantenimiento.

La diagnosis es una de las tareas de aplicación que mayor interés ha despertado en la comunidad investigadora de la IA. La diagnosis ha sido un área fundamental para el desarrollo de esta disciplina desde comienzos de los años 70, por motivos como:

- Presenta una buena combinación de aspectos teóricos y prácticos y su solución parece demandar inteligencia.
- Ha requerido, y permitido, experimentar con numerosas metodologías.
- Diversas técnicas y métodos desarrollados para la diagnosis se han aplicado a otras áreas o han dado lugar a nuevos campos de la IA. A modo de ejemplo, se citan los métodos desarrollados para el sistema experto MYCIN, paradigma de lo que actualmente se denomina Sistema Experto de Primera Generación y precursor de la moderna Ingeniería de Conocimiento. La diagnosis basada en modelos ha dado origen al paradigma de razonamiento denominado “razonamiento basado en modelos”, que a su vez ha sido un importante impulsor de la simulación cualitativa. De forma similar, el esfuerzo por formalizar las redes causales que utilizan muchos sistemas de diagnóstico ha cristalizado en las redes bayesianas.

La diagnosis se puede aplicar a sistemas de muy distinta naturaleza, como el cuerpo humano, automóviles o sistemas socioeconómicos. Hablaremos entonces de diagnosis médica, diagnosis de dispositivos físicos, etc.

Las características de los sistemas objeto de diagnóstico son tan dispares que no existe una metodología comúnmente aceptada para la realización del diagnóstico. Sin embargo, puede afirmarse que, generalmente, requiere un proceso donde razonamiento y actuación se entremezclan hasta encontrar el motivo por el cual un sistema se aparta del comportamiento deseado.

En este capítulo se van a presentar técnicas de diagnosis en las que los procesos de razonamiento tienen una formulación computacional y son realizados por un ordenador. Estas técnicas son el elemento básico de los sistemas de diagnosis, pero no el único. Aspectos relacionados con la interfaz humano-computador, especialmente importantes si el sistema se concibe como un asistente más que como un agente autónomo, o con la integración con los sistemas de información en los que se va a implantar el sistema, son esenciales en el diseño de un sistema de diagnosis. Pero estos tópicos requerirían otro capítulo por sí solos y no se van a abordar en este tema. La exposición se limitará a los principios básicos que consideramos que deben formar parte del bagaje de un ingeniero informático, con el objetivo de que comprenda los fundamentos de esta tecnología y con la esperanza de que le sirvan de punto de partida si quiere especializarse en el diseño de estos sistemas.

El resto del capítulo se organiza de la siguiente forma. Tras una introducción en la que se presenta parte de la terminología utilizada en la diagnosis, se propone un esquema genérico de sistema de diagnosis que servirá de marco de referencia para analizar y comparar distintos métodos y sistemas de diagnosis. A continuación, se estudiarán tres técnicas diferentes: árboles de fallos, métodos de clasificación simbólica y diagnosis basada en consistencia. El capítulo finaliza con una breve descripción de otras aproximaciones existentes, que no pueden ser tratadas en profundidad en este capítulo.

12.1.1 Algunas definiciones

Con el fin de precisar el alcance de esta tarea, vamos a proponer dos definiciones que, aunque similares en sus contenidos, aportan distintos matices.

Una primera definición. Davies define la diagnosis como “proceso de razonamiento y actuación para identificar las causas de un comportamiento anómalo para recuperar la funcionalidad deseada” [Davis, 1982]. Esta definición menciona distintos aspectos que se van a analizar a continuación.

La diagnosis como proceso. La definición hace hincapié en el carácter de proceso que tiene la diagnosis. Pone de manifiesto la necesidad de intercalar los procesos de razonamiento con la actuación sobre el sistema. Esta es una característica de especial interés en la diagnosis de dispositivos físicos. En ellos, parte del proceso de razonamiento está encaminado a la propuesta de nuevas observaciones o a la realización de pruebas sobre el comportamiento del dispositivo; de manera recíproca, el resultado de dichas pruebas u observaciones influirá en el proceso de razonamiento. Esta capacidad de actuación es más limitada si diagnosticamos sistemas biológicos, pero también

tiene su importancia. Así, cuando una dolencia nos hace acudir a la consulta médica, es muy frecuente que, antes de emitir un diagnóstico definitivo, el médico requiera la realización de algunas pruebas u observaciones adicionales: análisis clínicos, quizás una radiografía...

Diagnosis frente a verificación. También se desprende de esta definición que el proceso de diagnosis se inicia a partir de la detección de un comportamiento anómalo en un sistema. Cuando se comprueba el estado de un sistema en ausencia de un comportamiento anómalo, la tarea se denomina verificación o prueba [Stefik, 1995]. En el contexto de la industria de manufacturas, es habitual verificar que los distintos subsistemas y el dispositivo final satisfacen las especificaciones de diseño. En un entorno hospitalario, el equivalente sería el chequeo médico que se realiza a una persona en principio sana.

El concepto de causa. La definición también incluye la palabra causa. Qué se entiende por causa depende de varios factores: el sistema objeto de diagnóstico, el nivel de detalle o granularidad del diagnóstico, la capacidad de reparación, etc. En el campo de los dispositivos físicos, la causa puede ser un componente que no funciona de acuerdo a sus especificaciones, por ejemplo una válvula de regulación de flujo en una refinería. Se habla entonces de **localización** de fallos. Otras veces, el diagnóstico puede informar de cómo falla el dispositivo, por ejemplo indicando que la válvula se ha atascado y permanece totalmente abierta. Se habla entonces de **identificación** de fallos. En cualquier caso, la solución habitual consiste en sustituir el componente defectuoso, de modo que la localización puede ser suficiente. Estos conceptos no tienen un equivalente preciso en el campo de la medicina. La causa del malestar de un paciente se puede localizar en una úlcera de duodeno, pero esta información es insuficiente, pues no hay capacidad de sustituir el órgano afectado. La recuperación del paciente exige proponer una terapia adecuada y para ello hay que conocer si la úlcera tiene un origen vírico o psicosomático. A veces, la noción de causa está asociada a una explicación causal o cadena de causas-efectos. Por ejemplo, un depósito de almacenamiento de una planta de procesos puede presentar una fuga. El origen de la fuga puede deberse a que el depósito ha sufrido un impacto. En este caso, la reparación del depósito permite recuperar la funcionalidad del sistema. Pero no resulta difícil imaginar que la fuga del depósito pueda ser causada por un proceso de oxidación, debido a una pequeña pero continua fuga de un circuito de refrigeración próximo. En este tipo de sistemas, en que los fallos pueden propagarse entre subsistemas, una explicación causal, que encadene el fallo original con los restantes fallos que pueda generar, sería deseable.

Fallos internos y externos. En cualquier tipo de sistema, la causa del comportamiento anómalo se puede encontrar en el propio sistema o ser provocada por el entorno. Se distingue entonces entre fallos internos frente a fallos externos. Diremos que un fallo es interno cuando la causa del fallo se localiza en el interior del sistema: un componente averiado o una úlcera de duodeno. Diremos que un fallo es externo si alguna de las variables del entorno que interactúan con el sistema se aleja del rango que el sistema puede admitir. Por ejemplo, un motor diésel puede funcionar mal porque hemos llenado el depósito de gasolina sin plomo; ningún componente del sistema está fallando, pero el sistema manifiesta un comportamiento anómalo.

Si el motor no se ha deteriorado, bastaría con vaciar el depósito y añadir el combustible adecuado para recuperar el funcionamiento normal. En muchos problemas reales, los fallos externos son tan importantes como los internos, pues, además de provocar un comportamiento anómalo, pueden inducir fallos internos. Sin embargo, la mayoría de los sistemas de diagnóstico sólo consideran fallos internos.

Recuperar la funcionalidad deseada. El objetivo final de un sistema de diagnosis automático es facilitar que el sistema recupere la máxima funcionalidad posible. El diagnóstico es un medio que nos permitirá seleccionar las acciones más adecuadas para el fin buscado. El tipo de acción depende del sistema y del entorno de trabajo. En el caso de la medicina, las acciones son algún tipo de terapia: un tratamiento farmacológico, una intervención quirúrgica, una modificación de los hábitos del paciente o una combinación de ellos. En los dispositivos físicos, la recuperación suele requerir algún tipo de reparación. Si el dispositivo está formado por componentes interconectados, la reparación se limita a la sustitución del componente defectuoso. Aun en este caso, hay entornos en los que la reparación no es posible. Pensemos, por ejemplo, en un avión en vuelo o en el brazo de un robot trabajando sobre la superficie de Marte. En estos casos, es preciso diseñar el sistema de modo que las funcionalidades básicas del mismo se puedan mantener en presencia de algunos fallos mediante procedimientos de reconfiguración. También es posible diseñar los sistemas con componentes tolerantes a fallos.

Una segunda definición. Finalizaremos esta sección con una segunda definición debida a Console: “La diagnosis es la tarea que, dados un sistema y un conjunto de observaciones de un comportamiento anómalo, determina qué está mal en el sistema, con el fin de recuperar su funcionamiento correcto” [Console, 2000]. Obsérvese que, a pesar del tiempo transcurrido entre ambas definiciones, éstas son muy similares. Esta definición es interesante porque Console hace explícita la necesidad de disponer de observaciones.

La necesidad de observaciones. El hecho de que para diagnosticar un sistema sean necesarias observaciones del mismo puede parecer una obviedad. Pero la capacidad de diagnóstico depende fuertemente de las observaciones disponibles. Dado que la observación requiere algún tipo de medida que tiene un coste asociado, la mayor parte de los dispositivos proporcionan las observaciones mínimas para supervisar y gobernar el mismo. El diagnóstico suele requerir observaciones adicionales. En algunos sistemas, como los circuitos digitales, la obtención de observaciones adicionales no supone ningún problema: el coste de medir voltajes en distintos puntos de contacto es pequeño. Sin embargo, el no disponer de ellas de forma automática reduce la capacidad de diagnóstico. En otros sistemas, como el motor de un automóvil, el coste de medidas adicionales es elevado y sólo pueden obtenerse en talleres especializados. En ambos casos la capacidad de diagnóstico depende del diseño del dispositivo. Por ello, cada vez es más frecuente tener en cuenta la capacidad de diagnóstico, o diagnosibilidad, deseada en las etapas de diseño.

12.2 Elementos básicos de un sistema de diagnosis

Para facilitar la descripción de un sistema de diagnosis, comprender sus principios de funcionamiento y facilitar la comparación de distintos sistemas entre sí resulta de utilidad el esquema propuesto por Stefik [Stefik, 1995], y que está adaptado en la Figura 12.1.

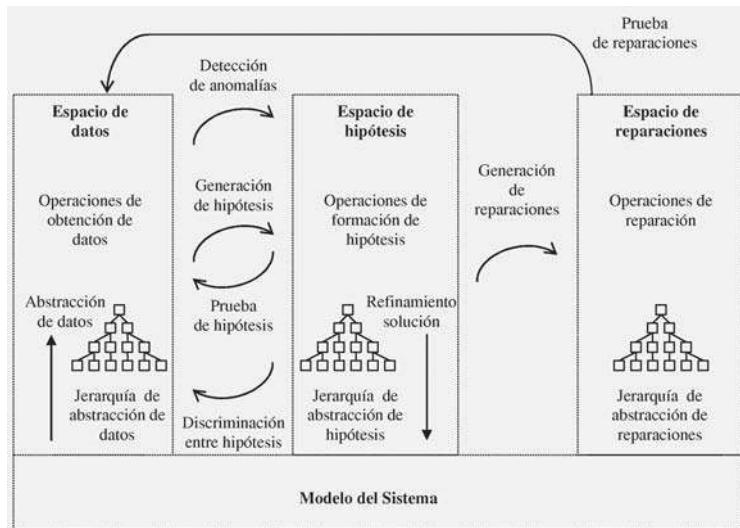


Figura 12.1: Esquema genérico de un sistema de diagnosis.

12.2.1 Espacio de búsqueda

De forma general, se puede concebir el problema de la diagnosis como un proceso que a partir de observaciones genera hipótesis capaces de explicar un funcionamiento anómalo para seleccionar las acciones de reparación. Por ello, es conveniente identificar un espacio de datos, un espacio de hipótesis y un espacio de reparaciones:

- **Espacio de datos.** El espacio de datos contiene un conjunto finito de posibles datos sobre el sistema y, quizás, su entorno. Estos datos pueden ser observaciones, parámetros, manifestaciones, síntomas, históricos, datos de laboratorio, hallazgos, etc.
- **Espacio de hipótesis.** El espacio de hipótesis incluye hipótesis de qué puede estar mal en el sistema. Una hipótesis puede agrupar numerosos fallos o describir un único fallo. Dependiendo del dominio, los fallos se pueden denominar causas, enfermedades, síndromes, etc.
- **Espacio de reparaciones.** El espacio de reparaciones incluye las acciones que se pueden realizar sobre el sistema para recuperar su funcionalidad.

En los tres espacios pueden darse distintos niveles de abstracción y organización en sus elementos. Así, en el espacio de datos se puede encontrar jerarquías de abstracción de datos que facilitan la transformación de las observaciones en conceptos que son más útiles para la diagnosis: la abstracción cualitativa del valor de la temperatura corporal al concepto de *fiebre* es un ejemplo de ello. De manera similar, una jerarquía de hipótesis permite pasar de una hipótesis más genérica, como septicemia, a una hipótesis más concreta, como septicemia linfática.

12.2.2 Modelo del sistema

El proceso de diagnosis necesita un modelo del sistema objeto de diagnóstico que permita establecer relaciones entre los datos, las hipótesis y las reparaciones. Los modelos pueden ser de muy distinta naturaleza, desde asociaciones entre datos e hipótesis hasta modelos detallados de la estructura y comportamiento del sistema. Con independencia de la naturaleza del modelo, este ha de soportar los procesos de razonamiento necesarios para explicar las discrepancias entre el comportamiento esperado y las observaciones del mismo.

El modelo del sistema puede solaparse con los espacios de búsqueda citados en la sección anterior, compartiendo estructuras comunes, como por ejemplo las jerarquías de diagnosis. Las operaciones que se realizan en los espacios pueden modificar el modelo de sistema, como activar el modelo de un fallo para intentar confirmarlo o descartarlo. Existe una gran variedad de modelos que se han utilizado para realizar la diagnosis. El tipo de modelo y el uso que se haga de él determinan las principales propiedades del sistema de diagnóstico.

12.2.3 Ejemplo de sistema a diagnosticar

Estas ideas se pueden ilustrar con un sistema sencillo cuyo esquema puede verse en la Figura 12.2. El sistema tiene tres componentes hardware, dos multiplicadores, M_1 y M_2 , y un sumador, A_1 . A , C , B y D son las entradas al sistema y F la salida, todas ellas observables. X e Y son puntos adicionales de medida. Los valores entre corchetes representan los valores de las observaciones. El espacio de datos para este sistema contiene los valores, observados y predichos, de las entradas y salidas de cada componente. La Figura 12.2 sólo muestra las observaciones en A , C , B , D y F .

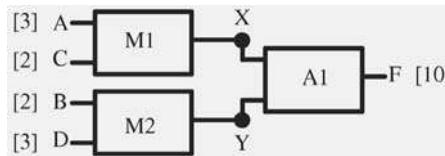


Figura 12.2: Ejemplo de sistema a diagnosticar.

Para simplificar la discusión se supondrá que sólo se está interesado en determinar qué componente está fallando: localización de fallos. Entonces, el espacio de hipótesis contiene las combinaciones de componentes que pueden fallar: $[M_1]$, $[M_2]$, $[A_1]$,

$[M_1, A_1]$, $[M_1, M_2]$, $[M_2, A_1]$ y $[M_1, M_2, A_1]$. En general, el número de hipótesis es de n^k , siendo n el número de comportamientos que puede mostrar un componente y k el número de componentes. En el caso de un problema de localización $n = 2$, que se corresponden al componente funcionando correcta o incorrectamente. Este ejemplo sencillo es suficiente para poner de manifiesto una de las principales dificultades de la diagnosis: el espacio de hipótesis crece exponencialmente con el número de componentes del sistema, k . La dificultad aumenta a medida que queremos un diagnóstico más preciso que nos informe de cómo está fallando el sistema, pues esto incrementa el valor de n .

Para este problema, el espacio de reparaciones contiene sólo tres acciones, que consisten en reemplazar el componente que falla por otro del mismo tipo que funcione correctamente.

En un sistema de este tipo, el modelo del sistema puede ofrecer información detallada sobre el dispositivo objeto de diagnóstico, en forma de estructura y comportamiento. La estructura está reflejada por las interconexiones entre los componentes, tal y como muestra la Figura 12.2. El comportamiento se describe proporcionando los modelos de los componentes. Para un sumador, el modelo de funcionamiento correcto indica que la salida es la suma de sus dos entradas. Para un multiplicador, la salida es el producto de sus dos entradas. Para un problema de localización, puede ser suficiente con proporcionar los modelos de funcionamiento correcto. Para la identificación de fallos también se necesitan modelos de comportamiento incorrecto, más difíciles de obtener.

Este tipo de modelos, basados en la estructura y el comportamiento, proporcionan mucha información sobre el dispositivo a diagnosticar y permiten la aplicación de métodos formales en el proceso de diagnóstico. Lamentablemente, no todos los sistemas pueden modelarse de esta forma.

12.2.4 Operaciones o subtareas

Cada uno de los espacios de búsqueda soporta operaciones, que se apoyan en el modelo del sistema. El espacio de datos soporta la obtención de datos u observaciones y, posiblemente, la abstracción de las mismas. Los datos pueden ser aportados al sistema a iniciativa de un agente externo (de forma síncrona o asíncrona) o pueden ser solicitados a iniciativa del sistema de diagnosis, de forma síncrona, si queremos vigilar un sistema periódicamente, o asíncrona, como parte del proceso de razonamiento, por ejemplo para discriminar entre hipótesis.

El espacio de hipótesis soporta las subtareas básicas del sistema de diagnosis, que se enumeran a continuación:

1. Monitorización.
2. Generación y prueba de hipótesis.
3. Discriminación de hipótesis.

Con frecuencia, se considera que la diagnosis sólo incluye las operaciones 2 y 3, esto es, generación y prueba de hipótesis y discriminación, separándola claramente de

la tarea de monitorización. Aunque esta división es apropiada en muchos sistemas, hay métodos de diagnosis en los que la monitorización y la generación y prueba de hipótesis están fuertemente acopladas, de modo que no se pueden diseñar de modo independiente.

12.2.4.1 Monitorización

También denominada detección de fallos en dominios técnicos. La monitorización es responsable de identificar las anomalías del sistema que se están diagnosticando. Generalmente, la monitorización es el primer paso del proceso de diagnóstico. La mayoría de los sistemas de diagnosis parten de la evidencia de un posible comportamiento anómalo para iniciar el proceso de diagnosis.

Dependiendo del modelo del sistema, la monitorización se puede limitar a comparar valores observados con una tabla de umbrales fijos, o puede requerir el uso de técnicas avanzadas de análisis y procesamiento de señales. En el primer caso, la tarea de monitorización es sencilla y el peso de la diagnosis recae en las tareas de generación y discriminación de hipótesis. Esta situación puede darse en un problema de diagnóstico médico, cuando el paciente acude a la consulta del médico de familia para comunicarle los síntomas iniciales, como presencia de mareos y cefaleas; esto da origen al proceso de diagnosis, que requerirá la obtención de más evidencia para la generación y discriminación de hipótesis. En el extremo opuesto, propio de sistemas dinámicos complejos cuyo comportamiento se intenta predecir mediante modelos dinámicos del mismo, el mayor peso de la diagnosis recae en la monitorización o detección de fallos, que se organiza de tal forma que las etapas de generación y discriminación de hipótesis se ven simplificadas.

Este tipo de monitorización se puede ilustrar con el sistema de la Figura 12.2. El ejemplo es sencillo porque está modelado a un nivel de abstracción en que el comportamiento de los componentes se describe mediante funciones aritméticas y es suficiente con considerar el comportamiento estacionario del mismo. Pero las ideas básicas son aplicables a sistemas más complejos. En este ejemplo, los valores de las entradas y la suposición de funcionamiento correcto de los multiplicadores permiten predecir los valores $X = 6$ e $Y = 6$. Suponiendo que el sumador también funciona correctamente, predecimos el valor $F = 12$. Este valor difiere de la observación $F = [10]$. El proceso de monitorización informaría de la anomalía y daría, además, una información útil para la diagnosis: si suponemos que no hay fallos en las observaciones, la anomalía sólo se puede deber a que uno de los tres componentes involucrados en la predicción no funciona correctamente.

12.2.4.2 Generación y prueba de hipótesis

Una vez detectada una anomalía, el siguiente paso es generar hipótesis que puedan explicarla y a continuación probar la viabilidad de las mismas, contrastándolas con las observaciones.

Los sistemas más sencillos utilizan la **hipótesis de fallo único**: se considera que nunca se producen dos o más fallos simultáneamente. Bajo la hipótesis de fallo único,

la operación de generación de hipótesis se puede reducir a una mera enumeración de los posibles fallos considerados por el sistema. Sin embargo, hay muchos sistemas en los que esta hipótesis no es razonable. En un problema de diagnóstico médico, generalmente interesaría diagnosticar todas las posibles enfermedades que sufre el paciente. Entonces hay que considerar la posibilidad de **fallos múltiples**, lo cual, como vimos en el ejemplo de la Figura 12.2, da lugar a un crecimiento exponencial del espacio de hipótesis. En presencia de fallos múltiples, la generación de hipótesis ha de ser más cuidadosa, utilizando las observaciones actuales para limitar el proceso de generación de hipótesis. Además, para reducir o simplificar la búsqueda, es deseable dotar de estructura al espacio de hipótesis. Muchos sistemas recurren a algún tipo de jerarquía de hipótesis. Una posibilidad es utilizar jerarquías de fallos que vayan de lo general a lo específico. Esta aproximación es particularmente útil en aquellos dominios donde ya existen estas jerarquías, como es el caso de la medicina. Otra posibilidad es crear jerarquías de componentes, agrupados por subcomponentes o por funcionalidades. Cualquiera que sea la naturaleza de estas jerarquías, permiten realizar al diagnóstico a distintos niveles de abstracción, de forma que cuando se baja de nivel de abstracción sólo se examinan las hipótesis cuyos antecesores se consideran posibles. Imagínese, por ejemplo, que se está diagnosticando un fallo en un ordenador digital. En el máximo nivel de abstracción, se puede considerar el sistema formado por una unidad central, un bus de memoria, un bus de periféricos y los distintos subsistemas conectados a estos buses. Si al máximo nivel de abstracción es posible localizar un fallo en el bus de memoria, podemos olvidarnos del resto de los componentes del sistema y focalizar el diagnóstico en el propio bus y sus conexiones.

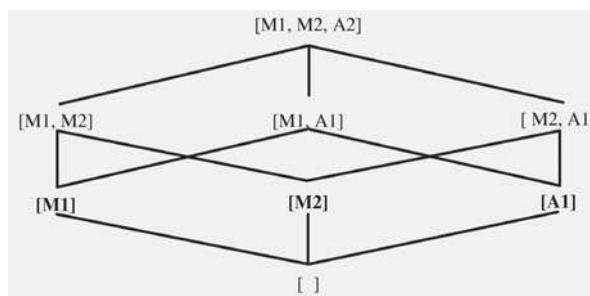


Figura 12.3: Retículo de hipótesis.

De nuevo, se utilizará el ejemplo de la Figura 12.2 para ilustrar estas ideas. En este caso, la tarea de monitorización, además de detectar las anomalías, nos proporciona información muy valiosa para la generación de hipótesis. Dado que la anomalía se ha detectado suponiendo que M_1 , M_2 y A_1 funcionan correctamente, es necesario que al menos uno de estos tres componentes falle. Por tanto, cualquier combinación de estos tres componentes es una hipótesis coherente con las observaciones. Una generación de hipótesis poco cuidadosa proporcionaría $[M_1]$, $[M_2]$, $[A_1]$, $[M_1, A_1]$, $[M_1, M_2]$, $[M_2, A_1]$ y $[M_1, M_2, A_1]$ como posibles causas de la anomalía. Pero, incluso en este caso sencillo, se puede simplificar la operación de generación sin más que dotar al espacio de

hipótesis de la estructura de retículo que induce la relación de inclusión, como muestra la Figura 12.3. Esto permite representar todas las hipótesis consideradas por las hipótesis minimales respecto a la relación de inclusión.

En este ejemplo todas las posibles hipótesis no vacías son consistentes con las observaciones. Como se aprecia en la Figura 12.3, las hipótesis minimales son $[M_1]$, $[M_2]$ y $[A_1]$, que representan de forma concisa las hipótesis actuales. Nótese que la hipótesis vacía, $[]$, no es consistente con las observaciones, porque si todos los componentes funcionasen correctamente el valor predicho de F debería ser igual que el valor observado.

La operación de prueba de hipótesis suele requerir contrastar las observaciones con las hipótesis generadas, rechazando aquellas que no son coherentes con las mismas. De alguna forma, el modelo de sistema debe relacionar los fallos con los síntomas observados, de manera que la presencia o ausencia de los mismos permita obtener información adicional sobre la validez de una hipótesis. Si el sistema utiliza la hipótesis de fallo único, y suponemos que un fallo genera siempre los mismos efectos, una simple enumeración de los síntomas asociados a cada causa puede ser suficiente. Vemos, pues, que el sistema de diagnosis más simple que podemos concebir, que denominaremos *generación y prueba exhaustivas*, es aquél que recurre a la hipótesis de fallo único y utiliza un modelo estático del sistema, consistente en un conjunto de asociaciones hipótesis-síntomas junto a los valores normales de algunas observaciones, necesarias para la monitorización. Si el espacio de hipótesis es suficientemente pequeño, bastará con generar sistemáticamente todas las hipótesis para podar a continuación aquellas cuyos síntomas no estén presentes.

Lamentablemente, pocos sistemas reales permiten una aproximación tan simple al proceso de diagnosis. Si el modelo del sistema no se reduce a asociaciones estáticas hipótesis-síntomas, el espacio de hipótesis es grande o no es razonable utilizar la hipótesis de fallo único, la diagnosis mediante generación y prueba exhaustivas no es viable.

En particular, la presencia de fallos múltiples complica el proceso de prueba de hipótesis debido a las interrelaciones entre síntomas de distintas causas. Estas interrelaciones son:

- Solapamiento de síntomas: los síntomas se solapan cuando dos fallos tienen síntomas comunes.
- Compensación de síntomas: este fenómeno se produce cuando dos fallos ejercen efectos contrarios sobre un mismo parámetro del sistema, de modo que síntomas que serían observables si los fallos se presentasen por separado no se manifiestan cuando se producen ambos fallos.
- Sinergia de síntomas: en este caso, dos fallos ejercen el mismo efecto sobre el mismo parámetro, de forma que síntomas ausentes cuando los fallos se producen por separado, sólo se manifiestan en presencia de ambos.

En aquellos sistemas dinámicos en los que alguna variable está gobernada por un sistema de control o regulación automática, se presenta un problema adicional:

- Enmascaramiento de síntomas: este fenómeno se produce cuando el sistema de regulación reacciona para contrarrestar el efecto de un fallo, de modo que un síntoma que sería apreciable en ausencia de regulación no se manifiesta.

Todos estos fenómenos dificultan extraordinariamente la prueba de hipótesis y exigen el uso de modelos del sistema más sofisticados. Debido a la imposibilidad de considerar todas las posibles combinaciones de causas, algunos modelos recurren a describir los síntomas de las combinaciones de fallos más probables. Otros recurren a algún mecanismo de gestión de certeza (probabilidad, lógica borrosa, factores de incertidumbre, ...), que se va acumulando a medida que se constata la presencia o ausencia de síntomas. También es posible modelar los fallos individualmente e inyectarlos en el modelo cuando se sospecha su presencia. En todos estos casos, es frecuente que los procesos de generación y prueba de hipótesis se intercalen.

Por último, tanto en sistemas dinámicos como estáticos, puede ocurrir que el comportamiento del sistema varíe con el tiempo y que los fallos no se manifiesten siempre de la misma forma. Un ejemplo de ellos son los **fallos intermitentes**, como los que se producen en una conexión en mal estado, que unas veces permite el paso de corriente y otras no. Debido a la dificultad de realizar el diagnóstico sobre este tipo de sistemas, es habitual suponer que el sistema no varía con el tiempo, que los fallos producen siempre los mismos síntomas, y que no se producen fallos intermitentes.

El ejemplo de sistema a diagnosticar introducido en la sección 12.2.3 ilustra otra aproximación al proceso de prueba de hipótesis. En este caso, la generación de hipótesis está restringida por el modelo del sistema y la forma en que se emplea para razonar, de modo que sólo se generan las hipótesis que son consistentes con las observaciones actuales. En consecuencia, el proceso de prueba no puede podar ninguna de las hipótesis generadas y se omite.

12.2.4.3 Discriminación de hipótesis

Como resultado del proceso de generación y prueba de hipótesis, se obtiene un conjunto de hipótesis, quizás ordenadas por alguna medida de certeza. Con el proceso de discriminación de hipótesis, se intenta proporcionar un diagnóstico más preciso reduciendo el número de las mismas.

¿Cómo reducir el número de hipótesis? La operación de generación y prueba de las mismas ya ha utilizado el modelo del sistema y las observaciones disponibles para generar las hipótesis plausibles. Con la información y el conocimiento disponible, poco más se puede hacer. El uso de métodos más sofisticados de razonamiento, sin inyectar más conocimiento o nuevas observaciones, no es de mucha ayuda. Por ello, la operación de discriminación recurre a la obtención de nuevas observaciones para reducir el número de hipótesis. Una primera cuestión es cómo vamos a utilizar las nuevas observaciones. Es cierto que la acumulación de evidencia a favor de una hipótesis puede favorecerla frente a las demás, pero si lo que queremos es un diagnóstico más preciso, una estrategia más adecuada es buscar evidencia que permita descartar hipótesis. En el caso ideal, la diagnosis finaliza cuando sólo nos queda una hipótesis y todas las alternativas han sido rechazadas. Por ello, la operación de discriminación suele trabajar buscando evidencia que permita rechazar hipótesis.

Otro motivo para ello es que la presencia de síntomas, por sí sola, no es suficiente para confirmar definitivamente la presencia de un fallo, pues varias hipótesis pueden generar los mismos síntomas. Por el contrario, la ausencia de síntomas que no se compensen ni enmascaren sí permite rechazar hipótesis.

Con algunas excepciones, la principal limitación de la tarea de diagnosis es la adquisición de observaciones, no los recursos de cómputo. Por regla general, no es posible, debido a los costes de todo tipo asociados a la adquisición de medidas, obtener todos los datos posibles para la realización del diagnóstico. Por ello, el aspecto más crítico en la etapa de discriminación de hipótesis es la utilización de una estrategia eficiente de obtención de datos.

Los nuevos datos se pueden obtener seleccionando nuevos puntos de prueba –en los que realizar nuevas observaciones–, proporcionando nuevos valores a las variables de entrada –vectores de entrada de prueba, que llevan al sistema a un estado diferente– o ejecutando procedimientos de prueba –que intentan verificar la funcionalidad de algún subsistema o componente. En el campo de la medicina, los nuevos puntos de prueba son similares a análisis clínicos o cualquier otro método de exploración no invasivo. Los vectores de entrada de prueba son análogos a algunas terapias, como la administración de medicamentos.

Una forma de seleccionar los nuevos puntos de prueba es utilizar la teoría de la información para identificar que nuevas observaciones pueden aportar más información al proceso de discriminación. Es una aproximación razonable en aquellos sistemas en que todas las observaciones tienen el mismo coste. Pero en muchos dominios de aplicación este criterio es insuficiente. A veces es preciso investigar primero las hipótesis asociadas con fallos más graves, como un problema en el sistema de frenado frente a una avería en el aire acondicionado de un automóvil. Algunas medidas que aportan mucha información pueden tardar tiempo en obtenerse, como puede ser un cultivo de una muestra biológica. Algunos costes son difíciles de evaluar, como el riesgo de someter a un paciente a una prueba agresiva. Si el sistema de diagnóstico obtiene información interactuando con el usuario, entonces será preciso considerar aspectos relacionados con la interacción humano-computador.

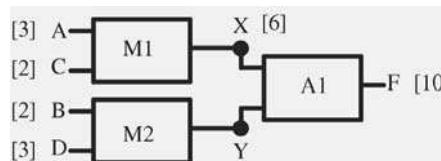


Figura 12.4: Obtención de una nueva observación.

Para finalizar esta sección, vamos a retomar el ejemplo de la Figura 12.2. La operación de generación y prueba de hipótesis nos ha proporcionado tres hipótesis minimales, $[M_1], [M_2], [A_2]$; bien entendido que cualquier hipótesis que contenga alguna de ellas también es una hipótesis válida. Dado que ya conocemos los valores de las entradas y la salida del sistema, los únicos nuevos puntos de prueba son X e Y. En este ejemplo, asumiendo que todas las medidas tienen igual coste, no hay ningún

criterio para preferir uno u otro. Supongamos que optamos por medir el valor de X y que el resultado es $X = [6]$, como indica la Figura 12.4.

Utilizando las observaciones $B = [2]$ y $D = [3]$, y suponiendo que M_2 funciona correctamente, predecimos $Y = 6$. Con la nueva observación y suponiendo funcionamiento correcto de A_1 , se predice $F = 12$. En consecuencia, al menos M_2 o A_1 han de fallar. ¿Qué podemos decir de M_1 ? Aunque la nueva observación nos puede hacer pensar que M_1 funciona correctamente, esta es una conclusión precipitada. Lo más que podemos decir es que, con las observaciones actuales, un fallo único en M_1 no es el responsable de la anomalía, pues esta se manifiesta con independencia de M_1 . La situación queda clara si examinamos el nuevo retículo de hipótesis, Figura 12.5, en el que se han tachado las hipótesis que no son consistentes con las observaciones.

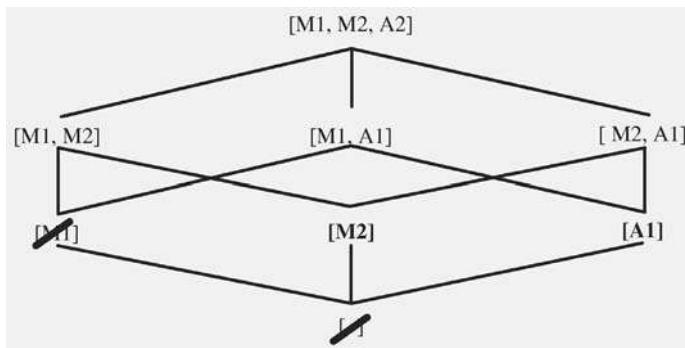


Figura 12.5: Hipótesis consistentes al observar $X = [6]$.

Las nuevas hipótesis minimales son $[M_2]$ y $[A_1]$. Es posible un fallo en M_1 —que no produce síntomas para las entradas consideradas— si se produce simultáneamente con un fallo en M_2 y/o A_1 . Ciertamente, esta circunstancia parece bastante improbable, pues exige que fallen dos componentes y que el fallo de M_1 no produzca síntomas con las observaciones actuales, pero no es imposible.

12.3 Diagnosis basada en árboles de fallos

Un árbol de fallos es un árbol de decisión cuyos nodos internos se etiquetan con pruebas, mientras que los nodos terminales contienen fallos o acciones de reparación. La Figura 12.6 muestra un ejemplo sencillo de árbol de fallos, adaptado de [Price, 1999]. Es un ejemplo muy simple, elegido porque es probable que resulte familiar a la mayoría de los lectores. Un árbol de fallos realista incluiría muchos más nodos y tendría que proporcionar muchos más detalles sobre la forma de realizar las pruebas y llevar a cabo las reparaciones.

Los nodos prueba son los responsables de obtener nuevas observaciones. Estas pueden limitarse a proponer nuevos puntos de medida (por ejemplo: ¿lucen los faros delanteros?) o pueden requerir la intervención sobre el sistema para comprobar alguna

condición (comprobar si hay chispas en una bujía puede exigir desmontarla del motor). Los nodos terminales identifican la causa de la anomalía y proponen una acción de reparación.

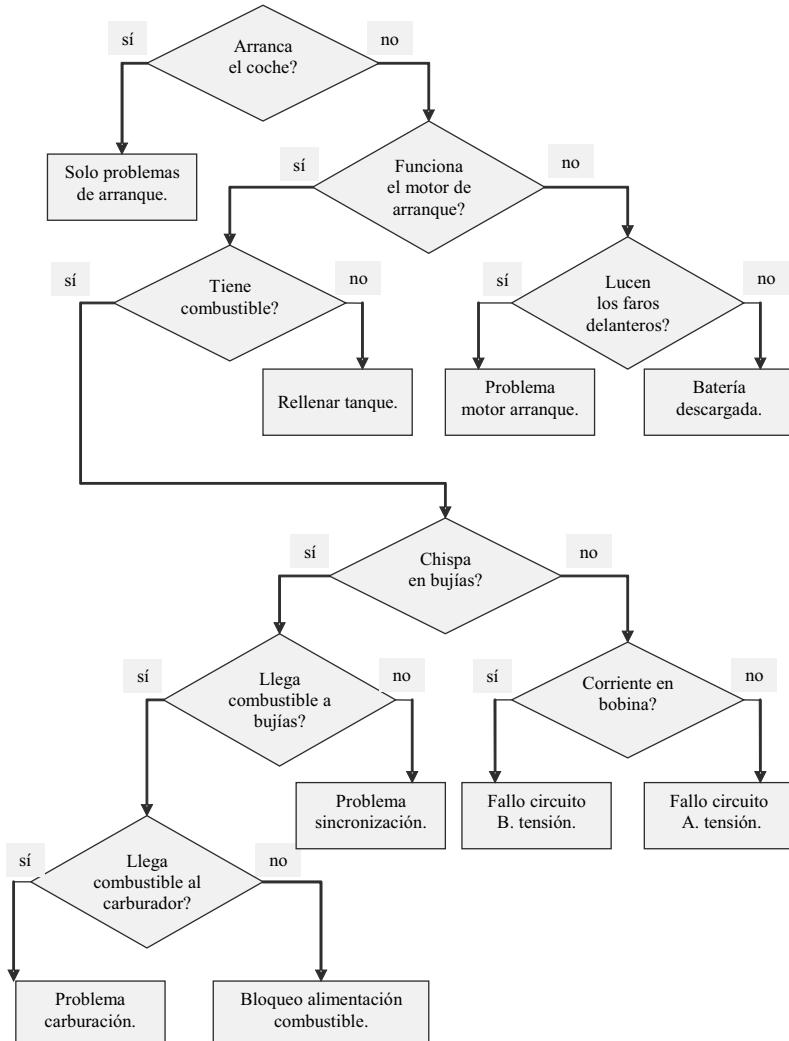


Figura 12.6: Árbol de fallos simplificado.

La tarea de diagnóstico se realiza recorriendo el árbol, comenzando por el nodo raíz, realizando las operaciones exigidas por los nodos prueba y avanzando por la rama etiquetada con el resultado de la prueba, hasta alcanzar un nodo terminal. En esencia, el árbol de fallos codifica la estrategia de diagnosis, indicándonos tras cada prueba cuál es el siguiente paso a seguir.

El desarrollo de un sistema basado en conocimiento a partir de un árbol de fallos puede abordarse utilizando una herramienta de desarrollo basada en reglas de producción. La aproximación más sencilla consiste en obtener una regla por cada camino del árbol desde el nodo raíz a los nodos terminales. El antecedente de la regla contendría la secuencia de nodos prueba, que se interpretaría como conjunción; el antecedente contendría el nodo terminal. Esta aproximación directa tiene el inconveniente de que requiere codificación adicional, que depende del intérprete de reglas, para garantizar que se mantiene la estrategia de solución. Una alternativa mejor consiste en introducir conceptos intermedios a medida que se recorren nodos prueba. Aunque se obtienen más reglas, la estrategia está explícita en ellas. La Figura 12.7 ilustra ambas aproximaciones.

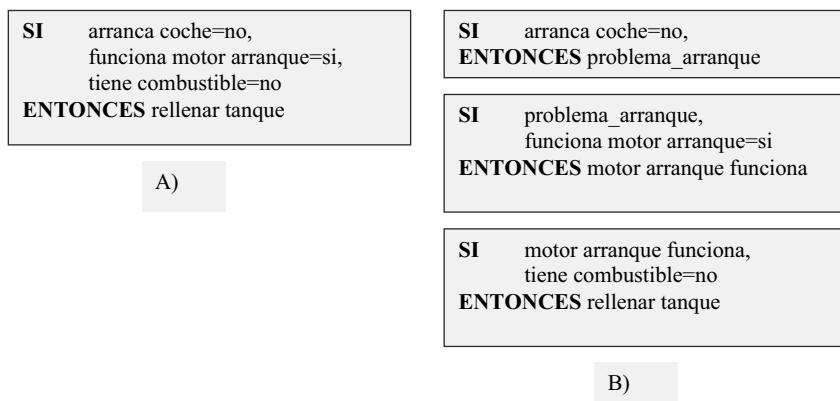


Figura 12.7: Ejemplo de reescritura del árbol de fallos mediante reglas. A) Aproximación directa. B) Introduciendo conceptos intermedios.

El desarrollo mediante una herramienta basada en reglas presenta varias ventajas sobre el uso de un lenguaje convencional: flexibilidad, pues es sencillo modificar el árbol cambiando las reglas; facilidades de explicación y traza de actividad proporcionadas por la herramienta y utilidades de interfaz. Sin embargo, el proceso manual de escritura de reglas es sensible a posibles fallos de codificación, más probables a medida que aumenta el tamaño del árbol. Por ello, se han desarrollado herramientas que permiten la edición gráfica del árbol de fallos, capaces de generar código ejecutable que también contiene facilidades de explicación e interfaz.

En el dominio de la ingeniería, los árboles de fallos son una herramienta habitual para documentar los procesos de diagnosis y reparación en los manuales de mantenimiento. Algo similar ocurre en el campo de la medicina, donde existen planes y protocolos específicos para diagnosis y tratamientos. Ello explica por qué gran cantidad de sistemas basados en conocimiento utilizan esta aproximación al diagnóstico: es un método de diagnosis que reproduce la estrategia con la que los profesionales están familiarizados. Para estos especialistas, un sistema de diagnóstico basado en árboles de fallos proporciona un soporte computacional a las tareas que realizan habitualmente.

Si, además, proporciona facilidades de edición gráfica del árbol de fallos, facilita enormemente la creación, actualización y distribución de los árboles de fallos. Con estos sistemas, esta tarea puede ser realizada por los propios médicos o ingenieros.

Para comprender las ventajas y los inconvenientes de los sistemas de diagnosis basados en árboles de fallos, vamos a adaptar el esquema genérico propuesto en la sección 12.2.2 a este tipo de sistemas, como resume la Figura 12.8.

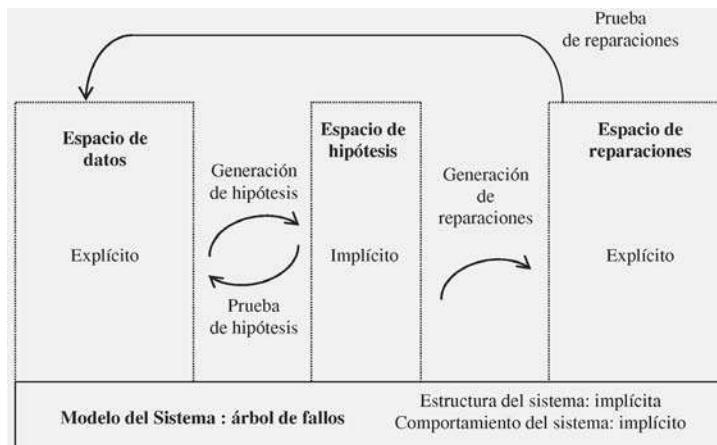


Figura 12.8: Esquema particularizado para los sistemas de diagnosis basados en árboles de fallos.

En estos sistemas, los nodos prueba mantienen representaciones explícitas de las observaciones y las pruebas. De manera similar, los nodos terminales representan explícitamente el espacio de reparaciones, indicando las acciones de reparación. El espacio de hipótesis se manipula de forma implícita, como consecuencia de atravesar los nodos del árbol. El modelo del sistema no proporciona información sobre la estructura o comportamiento del sistema objeto de diagnóstico: sólo incluye la estrategia de solución, que utiliza esta información de forma implícita. Como consecuencia, los árboles de fallos son difíciles de elaborar y de mantener: pequeños cambios en el dispositivo a diagnosticar pueden requerir reelaborar gran parte del árbol, pues no es fácil determinar cómo este se ve afectado por dichos cambios; incluso simples reordenaciones de las pruebas pueden requerir cambios importantes en el árbol, pues la interpretación de estas pruebas depende del contexto en que se realizan. La ausencia de modelos de comportamiento y la manipulación implícita de las hipótesis de diagnosis también dificulta la generación de explicaciones de diagnosis, más allá de reproducir el camino recorrido en el árbol. Este tipo de sistemas tampoco permite la detección de anomalías, pues ni siquiera dispone de un modelo de funcionamiento correcto para las condiciones de funcionamiento normal (por ejemplo, una simple tabla de valores esperados).

12.4 Diagnosis basada en modelos de clasificación simbólica

La tarea de clasificación se caracteriza por seleccionar una clase entre un conjunto finito –y habitualmente pequeño– de posibles clases, conociendo las características de las clases y las del objeto a clasificar.

Un gran número de problemas del mundo real se puede abordar mediante métodos de clasificación. Ejemplos bien conocidos son la clasificación taxonómica de seres vivos en familias y especies o la clasificación de objetos celestes a partir de imágenes de satélites. En general, aquellas tareas que se puedan solucionar sin necesidad de construir nuevas soluciones, sino eligiendo una entre un catálogo de posibles soluciones ya conocidas, son susceptibles de ser abordadas como una tarea de clasificación. La diagnosis también se puede abordar como una tarea de clasificación. Las clases serían los posibles fallos y el objeto a clasificar el sistema objeto de diagnosis. Si el conjunto de posibles fallos es bien conocido y estos se pueden caracterizar a partir de las observaciones del sistema, plantear la diagnosis como una tarea de clasificación es razonable. Si, por el contrario, la naturaleza de los fallos es desconocida (fallos difíciles de caracterizar, dispositivos novedosos, ...) o se presentan fallos múltiples que no se manifiestan como una mera agregación de los síntomas de los fallos individuales, la tarea de clasificación no es la más adecuada para realizar la diagnosis.

Una ventaja adicional de la tarea de clasificación es su sencillez: tanto los modelos de clasificación como los métodos que los utilizan son conceptualmente simples. Esto explica, quizás, la popularidad de los mismos: gran parte de los Sistemas Expertos de Primera Generación que abordan problemas de diagnóstico lo enfocan como un problema de clasificación. Sistema como MYCIN, INTERNIST o CENTAUR realizan una tarea de clasificación. Todos estos ejemplos pertenecen al dominio de la medicina, en el cual frecuentemente sólo se dispone de modelos de clasificación obtenidos por vía experimental. Pero también hay ejemplos de otros dominios: MORE –extracción de hidrocarburos–; MUD –condiciones de fluidos en perforaciones–, o SOPHIE –sistema pionero en la detección de fallos en circuitos electrónicos.

En esta sección comenzamos presentando un modelo de clasificación simple, que ayudará a comprender la naturaleza de esta tarea. Seguidamente se amplía este modelo, para llegar a la clasificación jerárquica, base de un gran número de sistemas de diagnosis.

12.4.1 Clasificación simple

Vamos a introducir los métodos de clasificación simbólica para la diagnosis con un modelo de recubrimiento finito, conceptualmente sencillo, pero que pone de manifiesto las características relevantes de la clasificación cuando se utiliza para resolver un problema de diagnosis.

12.4.1.1 Modelo de clasificación por recubrimiento

Un modelo de clasificación por recubrimiento es una quíntupla $\langle D, S, C^+, C^-, OBS \rangle$, siendo:

- D , Espacio de Datos, un conjunto finito de datos $\{D_1, D_2, \dots, D_k\}$.
- S , Espacio de Soluciones, un conjunto finito de soluciones $\{S_1, S_2, \dots, S_m\}$ o candidatos.
- C^+, C^- , son Relaciones de Recubrimiento, $C^+ \subset S \times D, C^- \subset S \times D$, con $C^+ \cap C^- = \emptyset$.
- OBS es el conjunto de valores que toman los datos D_i de D .

Los elementos de D representan las observaciones que se pueden realizar directamente sobre el sistema y toman valores en el conjunto $\{0, 1, ?\}$ indicando su ausencia, su presencia o la no disponibilidad de la observación. Por ejemplo, si D_i representa el síntoma “fiebre” y se observa que el paciente no tiene fiebre, en el conjunto OBS se tendrá $D_i = 0$. En el sistema de dos multiplicadores y un sumador introducido en la sección 12.2.3 el conjunto D es $\{A = 3, B = 2, C = 2, D = 3, X = 6, Y = 6, F = 12\}$. El conjunto de observaciones iniciales es $OBS = \langle 1, 1, 1, 1, ?, ?, 0 \rangle$.

Los elementos de S , candidatos, representan posibles soluciones del problema, por ejemplo bronquitis o asma. En el sistema de dos multiplicadores y un sumador, $S = \{M_1, M_2, A_1\}$, indicando que cada uno de los componentes individuales puede fallar.

Los elementos de C^+ y C^- son pares (S_j, D_i) que representan condiciones necesarias para la consistencia de una solución con un conjunto de observaciones. La combinación de C^+ y C^- define un patrón de observaciones necesarias para la consistencia de un candidato con los datos. Para cada par (S_j, D_i) sólo puede darse uno de los siguientes casos:

- $(S_j, D_i) \in C^+$. Se interpreta cada par $(S_j, D_i) \in C^+$ diciendo que el candidato S_j es consistente con la observación $D_i = 1$; en otras palabras, si $D_i = 0, S_j$ no puede ser solución.
- $(S_j, D_i) \in C^-$. Se interpreta cada par $(S_j, D_i) \in C^-$ diciendo que el candidato S_j es consistente con la observación $D_i = 0$; en otras palabras, si $D_i = 1, S_j$ no puede ser solución.
- $(S_j, D_i) \notin C^+, (S_j, D_i) \notin C^-$. Se llegará a la conclusión de que el dato D_i es irrelevante para la solución S_j . En cualquier otro caso, el dato D_i es relevante para la solución S_j .

Una posible solución, S_j , es **consistente** con un conjunto de observaciones OBS si y sólo si S_j es consistente con todas las observaciones disponibles; en caso contrario, S_j es inconsistente. Diremos que un candidato es una **explicación** de OBS si y sólo si S_j es consistente con las observaciones de todos sus datos relevantes. Este modelo de clasificación proporciona de forma explícita un mecanismo para rechazar hipótesis

de fallo: los candidatos inconsistentes pueden eliminarse del conjunto de hipótesis de diagnosis, pues no pueden ser solución del problema. El modelo básico no define de forma explícita cuál es la solución del modelo de clasificación: es preciso introducir criterios adicionales, que pueden depender del dominio de aplicación.

12.4.1.2 Ejemplo de modelos de clasificación por recubrimiento

Una vez definidas las relaciones C^+ y C^- , cada patrón de observaciones determina qué candidatos son rechazados, cuáles son consistentes y cuáles son una explicación. En el caso ideal, el patrón de observaciones permite rechazar todos los candidatos excepto uno, que además es una explicación. Sin embargo, pueden presentarse todas las posibles combinaciones: desde un patrón que rechaza todos los candidatos hasta un patrón que no rechaza ninguno, con varios candidatos consistentes y varias explicaciones. Para examinar esta variedad de posibilidades, es útil representar gráficamente la estructura de las relaciones C^+ y C^- , el patrón de observaciones y el estado de los candidatos correspondiente a cada patrón, como ilustra la Figura 12.9, adaptada de [Stefik, 1995]. Los pares $(S_j, D_i) \in C^+$ se representan mediante líneas continuas; los pares $(S_j, D_i) \in C^-$ mediante líneas discontinuas.

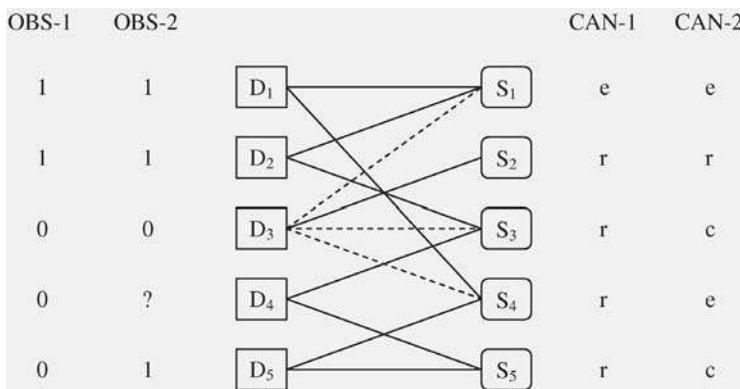


Figura 12.9: Representación gráfica de un modelo de clasificación por recubrimiento.

El patrón de observaciones $OBS-1$ es un ejemplo del caso ideal: S_1 es una explicación, porque es consistente con todas sus observaciones relevantes, y todos los demás candidatos son inconsistentes, por lo que son rechazados. Con el patrón de observaciones $OBS-2 = \langle 1, 1, 0, ?, 1 \rangle$ tenemos:

- Dos explicaciones, S_1 y S_4 , pues disponemos de todas sus observaciones relevantes y son consistentes con todas ellas.
- Dos soluciones consistentes con las observaciones disponibles, S_3 y S_5 , pero que no son explicaciones porque no disponemos de la observación de D_4 , relevante para ambas.
- Una solución rechazada, S_2 , pues no es consistente con la observación $D_3 = 0$.

Podemos utilizar este tipo de modelos con el ejemplo de la Figura 12.2. Una formulación sencilla de este problema, típica de los modelos de clasificación, modelaría el funcionamiento correcto de los componentes de este sistema para un punto de trabajo concreto, que vendría definido por el conjunto de datos. La Figura 12.10 muestra el modelo resultante. Sabiendo que $D = \{A = 3, B = 2, C = 2, D = 3, X = 6, Y = 6, F = 12\}$, que se corresponde con el funcionamiento correcto del sistema, el modelo simplemente refleja que un componente falla cuando la salida observada del componente no se corresponde con la esperada.

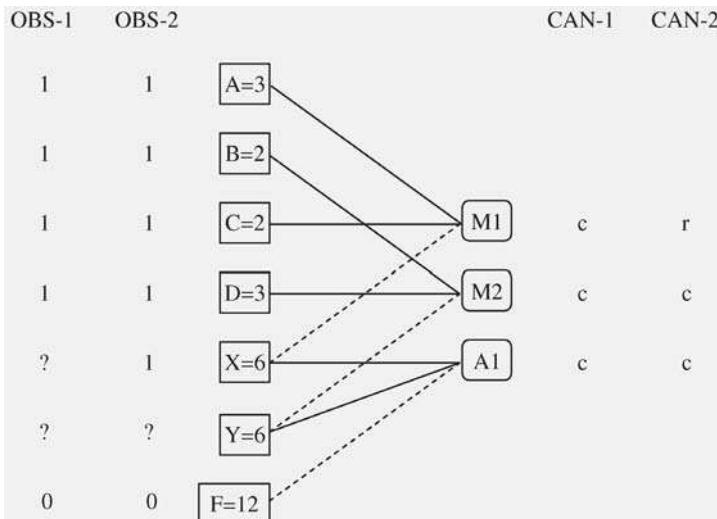


Figura 12.10: Modelo de recubrimiento para el sistema de dos sumadores y un multiplicador.

A las observaciones iniciales, $\{A = [3], B = [2], C = [2], D = [3], F = [10]\}$, le corresponde el patrón $OBS-1$, que da lugar a tres candidatos consistentes. La observación adicional $X = [6]$ proporciona el patrón $OBS-2$, que permite rechazar M_1 . De nuevo, necesitamos criterios adicionales para determinar que es una solución al problema.

12.4.1.3 Soluciones de un problema de clasificación por recubrimiento

Los modelos de clasificación por recubrimiento permiten rechazar candidatos, identificar cuáles son consistentes y cuáles explicaciones. Qué constituye una solución al problema de clasificación y, por tanto, al de diagnosis, depende de requisitos adicionales que se pueden imponer a los candidatos no rechazados. La naturaleza de estos requisitos es variada, y nos limitaremos a presentar algunos de los más comunes, agrupándolos en tres dimensiones:

- Hipótesis de fallo único frente a fallo múltiple.
 - Hipótesis de fallo único. Bajo esta hipótesis, las soluciones son candidatos del espacio de soluciones. Pueden darse dos casos:
 - * Soluciones mutuamente excluyentes: la solución sólo contiene un único candidato. En este caso, la primera explicación encontrada permite descartar todas las demás.
 - * Las soluciones no son mutuamente excluyentes. Si las soluciones no son mutuamente excluyentes, puede haber más de una clase solución. En este caso, la solución contendrá un conjunto de candidatos del espacio de soluciones, cuya interpretación es que cualquiera de ellos, por sí solo, soluciona el problema.
 - Hipótesis de fallo múltiple. En este caso, los candidatos del espacio de soluciones se combinan para formar una única solución compuesta. La interpretación es que están presentes todos los fallos incluidos en la solución compuesta.
- Criterio de inclusión.
 - Consistencia: se incluyen todas las clases consistentes con algún dato observado.
 - Conservadora: se incluyen todas las soluciones consistentes, aunque no se disponga de ninguna de sus observaciones relevantes.
 - Explicación: sólo se incluyen las soluciones que son explicaciones.
 - Completa: la solución ha de explicar todas las observaciones.
 - Por eliminación: sólo se acepta una solución cuando todas las demás han sido rechazadas.
- Preferencia de hipótesis.
 - Evidencia: preferir las soluciones consistentes con más datos.
 - Minimalidad: definir la solución como los subconjuntos minimales de las soluciones compuestas.
 - Ponderación: utilizar alguna función adicional para priorizar las hipótesis. Por ejemplo, probabilidad a priori, a posteriori o alguna medida de certeza.

La presencia de fallos múltiples complica sustancialmente el modelo básico. Si se han de considerar todas las posibles combinaciones, el modelo de recubrimiento es insuficiente, debido a la explosión combinatoria y a las interrelaciones entre los síntomas cuando hay más de un fallo. Si suponemos independencia entre los síntomas –esto es, sin compensación, sinergia ni enmascaramiento–, y consideramos un número reducido de combinaciones –por ejemplo, limitándonos a fallo doble–, la composición de soluciones se puede abordar con el modelo básico: basta con combinar los candidatos simples que se acepten como solución.

El efecto de estos criterios se puede ilustrar con el ejemplo de los multiplicadores y el sumador. Con el patrón de observaciones *OBS-2* (véase la Figura 12.10), hay dos candidatos consistentes, a pesar de que se desconoce el valor de Y, dato relevante para ambos. Si exigimos que la solución sea una explicación, el problema no tendría solución para este patrón de observaciones. Nos conformaremos, pues, con exigir que las soluciones sean consistentes.

- Hipótesis de fallo único. Solución: $\{M_2, A_1\}$. Es decir, falla M_2 o falla A_1 , no ambos a la vez.
- Fallo múltiple. Solución: $\{M_2, A_1, [M_2, A_1]\}$. En este caso, falla M_2 , o falla A_1 o ambos a la vez.
- Fallo múltiple, criterio de minimalidad. Solución $\{M_2, A_1\}$. Aunque la solución es más compacta, el significado es el mismo que en el caso precedente. Simplemente se omite el candidato $[M_2, A_1]$ porque no es minimal.

12.4.1.4 Clasificación simple mediante generación y prueba exhaustivas

Vamos a introducir los métodos de diagnosis para modelos de clasificación por recubrimiento con un método muy sencillo, que denominaremos generación y prueba exhaustivas, descrito en el Algoritmo 12.1. Se trata de un método demasiado simple para ser de utilidad en un problema real de diagnosis, pero permite ilustrar el funcionamiento básico de estos métodos. El método se basa en tres suposiciones:

- Todas las observaciones necesarias están disponibles al principio del proceso.
- El espacio de soluciones, S, es lo suficientemente pequeño como para poder considerar cada candidato individualmente.
- Se utiliza la hipótesis de fallo único.

La primera suposición indica que el método no dispone de ningún mecanismo para obtener nuevas observaciones: o se dispone inicialmente de todas las observaciones necesarias, o no se podrá refinar más la solución discriminando hipótesis con nuevas observaciones. La segunda suposición indica que las soluciones son los candidatos del espacio de soluciones, evitando la explosión combinatoria de soluciones compuestas. La última suposición recuerda las limitaciones de los métodos de búsqueda exhaustiva.

Algoritmo 12.1 Diagnosis mediante generación y prueba exhaustivas.

```
Soluciones ←  $\emptyset$ 
obtener OBS
para cada Candidato ∈ S hacer
    si Prueba(Candidato) entonces
        Soluciones ← Soluciones  $\cup$  Candidato
    fin si
fin para
devolver (Soluciones)
```

El método de generación y prueba exhaustivas simplemente obtiene las observaciones disponibles, considera que todos los candidatos del espacio de soluciones son hipótesis de diagnóstico válidas y utiliza el procedimiento prueba para rechazar los candidatos no consistentes con las observaciones actuales.

Se deja al lector el ejercicio sencillo de comprobar el funcionamiento de este método sobre el ejemplo de sistema a diagnosticar introducido en la sección 12.2.3 y cuyo modelo de recubrimiento se describe en la Figura 12.10. Suponer que el patrón de observaciones es $OBS-3 = \langle 1, 1, 1, 1, 1, 0, 0 \rangle$.

Si se compara este método con el esquema genérico de sistema de diagnosis introducido en la sección 12.2 vemos que:

- El modelo del sistema es el modelo de clasificación por recubrimiento, $\langle D, S, C^+, C^-, OBS \rangle$, que no representa de forma explícita ni la estructura ni el comportamiento del mismo. Sólo se dispone de forma explícita de las relaciones entre síntomas y fallos, dadas por C^+ y C^- .
- El espacio de datos es D , el de hipótesis es S y no se utiliza el espacio de reparaciones. Este tipo de modelos sólo razonan con las hipótesis de diagnóstico y los datos.

En relación con las operaciones básicas:

- La monitorización no es explícita. Simplemente se supone que hay alguna anomalía en el sistema.
- La operación de generar hipótesis se reduce a considerar, sucesivamente, cada candidato como hipótesis de diagnóstico.
- La operación de prueba de hipótesis, soportada por el procedimiento prueba, se limita a utilizar las relaciones de recubrimiento del modelo de clasificación para eliminar los candidatos inconsistentes con las observaciones.
- No hay etapa de discriminación de hipótesis.

12.4.1.5 Clasificación simple mediante generación guiada por datos

La eficacia computacional del método de generación y prueba exhaustiva puede mejorar si la generación de candidatos se realiza a partir de un número reducido de observaciones. En los problemas de diagnosis, es habitual que la subtarea de monitorización se encargue de obtener los valores de un conjunto reducido de observaciones del sistema. Estas observaciones se seleccionan según su capacidad potencial de detectar anomalías y de discriminar candidatos. Típicamente, la subtarea de monitorización sólo devuelve los valores de aquellas observaciones que indican la presencia de una anomalía para la que puede existir alguna solución consistente con dicha observación. El método se basa en las siguientes suposiciones:

- Disponemos de un procedimiento, *Monitor*, que obtiene el valor de un conjunto reducido de observaciones y nos devuelve en $OBS_{anómalas}$ las observaciones para las que existen candidatos consistentes. Este conjunto de candidatos es mucho menor que el conjunto de posibles soluciones.
- Todas las observaciones necesarias están disponibles al principio del proceso.
- Disponemos de un procedimiento, *GenerarCandidatos*, capaz de generar eficientemente las soluciones que son consistentes con $OBS_{anómalas}$.
- Se utiliza la hipótesis de fallo único.

Algoritmo 12.2 Diagnosis mediante generación guiada por datos.

```
Soluciones ← ∅
OBSanómalas = Monitor()
si OBSanómalas ≠ ∅ entonces
    obtener OBS
    Candidatos ← GenerarCandidatos(OBSAnómalas)
    para cada Candidato ∈ Candidatos hacer
        si Prueba(Candidato) entonces
            Soluciones ← Soluciones ∪ Candidato
        fin si
    fin para
fin si
devolver (Soluciones)
```

El método de generación guiada por datos comienza llamando al procedimiento *Monitor* para obtener $OBS_{anómalas}$; sólo en el caso de que alguna observación indique la presencia de alguna anomalía se procede con la subtarea de diagnosis. Esta comienza obteniendo todas las observaciones disponibles. A diferencia de la generación y prueba exhaustivas, este método se limita a generar las hipótesis de diagnosis que son consistentes con $OBS_{anómalas}$, mediante el procedimiento *GenerarCandidatos*, lo que puede reducir de forma drástica el número de hipótesis generadas. Finalmente, el procedimiento *prueba* rechaza las hipótesis que no son consistentes con las observaciones actuales.

Veamos cómo podemos utilizar este método con nuestro ejemplo de sistema a diagnosticar introducido en 12.2.3 y cuyo modelo de recubrimiento se describe en la Figura 12.10. En este caso, una diseño razonable sería aquél en el cual el procedimiento *Monitor* se limitara a obtener las observaciones en las salidas de los componentes individuales: X, Y, F . Supongamos que al observar el sistema se obtiene: $A = [3]$, $B = [2]$, $C = [2]$, $D = [3]$, $X = [6]$, $Y = [5]$, $F = [10]$. El patrón de observaciones correspondientes es $OBS-3 = \langle 1, 1, 1, 1, 1, 0, 0 \rangle$. El procedimiento *Monitor* busca los valores actuales de X, Y, Z y devuelve los elementos del patrón para los que pueden existir soluciones consistentes: $OBS-3.6=0$ y $OBS-3.7=0$ —donde la notación $OBS-3.X$ denota el elemento X del patrón $OBS-3$. *GenerarCandidatos* obtiene $\{M_2, A_1\}$: M_2 es consistente con $OBS-3.6=0$ y A_1 es consistente con $OBS-3.7=0$. El procedimiento *prueba*, considerando $OBS-3$, rechaza A_1 , pues no es consistente con $OBS-3.6=0$.

Si se compara este método con el esquema genérico de sistema de diagnóstico introducido en la sección 12.2 vemos que:

- El modelo del sistema es el modelo de clasificación por recubrimiento, $\langle D, S, C^+, C^-, OBS \rangle$, que no representa de forma explícita ni la estructura ni el comportamiento del mismo. Sólo se dispone de forma explícita de las relaciones entre síntomas y fallos, dados por C^+ y C^- .
- El espacio de datos es D , el de hipótesis es S , y no se utiliza el espacio de reparaciones.

En relación con las operaciones básicas:

- La subtarea de monitorización es explícita, encargándose de detectar un comportamiento anómalo del sistema.
- La operación de generar hipótesis se reduce a obtener aquellos candidatos que son consistentes con las observaciones anómalas obtenidas por la monitorización.
- La operación de prueba de hipótesis, soportada por el procedimiento *prueba*, se limita a utilizar las relaciones de recubrimiento del modelo de clasificación para eliminar los candidatos inconsistentes con las observaciones.
- No hay etapa de discriminación de hipótesis.

Una última aclaración sobre el ejemplo con el que hemos ilustrado el método. Con las observaciones disponibles, y razonando sobre el sistema como hicimos en la sección 12.2, se habría obtenido dos candidatos a fallo único: $\{M_2\}$ y $\{A_1\}$. Ahora, sin embargo, sólo se ha obtenido $\{M_2\}$. Esto se debe a que el método de clasificación utiliza un modelo de recubrimiento que únicamente refleja relaciones entre síntomas y causas. El modelo de recubrimiento sólo permite capturar estas relaciones, de forma natural, en torno a un punto de funcionamiento. La presencia de un fallo en M_2 modifica el punto de funcionamiento de A_1 y el sistema no puede diagnosticar el fallo de este componente.

12.4.2 Clasificación jerárquica

La clasificación jerárquica, también denominada clasificación heurística, es una variante de la clasificación simple que añade al modelo de clasificación por recubrimiento dos elementos adicionales: una jerarquía de abstracción de datos y una jerarquía de refinamiento de soluciones. La Figura 12.11 muestra un modelo de recubrimiento para la clasificación jerárquica. Nótese que el modelo de recubrimiento $\langle D, S, C^+, C^-, OBS \rangle$ no se modifica: simplemente se dota de estructura el espacio de soluciones y se añade una etapa de abstracción de datos.

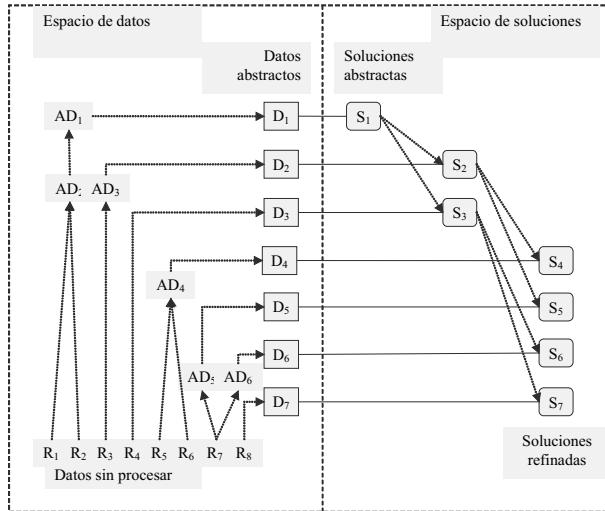


Figura 12.11: Clasificación jerárquica: modelo de recubrimiento más jerarquía de abstracción de datos y jerarquía de refinamiento de soluciones.

12.4.2.1 Jerarquía de abstracción de datos

La jerarquía de abstracción de datos describe cómo procesar los datos básicos para obtener conceptos que estén al nivel de abstracción adecuado para caracterizar las soluciones. Clancey propone tres tipos básicos de abstracciones, que ilustra en el dominio médico, tal y como se usa en MYCIN, véase la Figura 12.16:

- Abstracción Cualitativa: “Recuento leucocitario de 2500” se abstraerá a “Recuento leucocitario bajo”.
- Abstracción por Definición: “Recuento leucocitario bajo” se abstraerá a “Leucopenia”, pues así se define la leucopenia.
- Abstracción por Generalización: “Leucopenia” se abstraerá a “Inmunodepresión”, pues la leucopenia es un tipo de inmunodepresión.

Otros tipos de abstracción son posibles. Por ejemplo, se pueden combinar las lecturas de dos sensores de presión en un único dato abstracto: “presión alta”. En general, son numerosas las transformaciones que pueden aplicarse a los datos para obtener conceptos de interés para la diagnosis.

Es importante notar que el proceso de abstracción de datos es, a su vez, una forma de clasificación cuyo resultado, las clases abstractas, sigue perteneciendo al espacio de datos. Tanto las clases abstractas como los datos básicos pueden usarse como datos del modelo de recubrimiento para realizar la clasificación en el espacio de soluciones.

12.4.2.2 Jerarquía de refinamiento de soluciones

La jerarquía de refinamiento de soluciones permite considerar clases de soluciones que se van refinando a medida que se dispone de nuevos datos. El espacio de soluciones de la Figura 12.11 ilustra una jerarquía sencilla donde cada subclase se diferencia de su superclase padre porque requiere un dato adicional para determinar su consistencia. Con esta estructura, cualquier patrón de observaciones que haga consistente a S_7 , también hace consistente a S_3 y a S_1 . Para el patrón de observaciones (de los datos abstractos) $\langle 1, ?, ?, ?, ?, ?, ? \rangle$, todas las clases son consistentes, lo que se puede resumir diciendo que S_1 es consistente. La topología de la jerarquía permite eliminar aquellos candidatos sucesores de un nodo inconsistente. El patrón $\langle 0, ?, ?, ?, ?, ?, ? \rangle$ es inconsistente con todas las clases, lo que se resume diciendo que S_1 es inconsistente. El patrón $\langle 1, 0, ?, ?, ?, ?, ? \rangle$ es consistente con S_1 , S_3 y todos los sucesores de S_3 , pero inconsistente con S_2 y todos sus sucesores.

Denominaremos **eliminación jerárquica** a este método de eliminación de candidatos. La única condición necesaria para aplicar la eliminación jerárquica es que el conjunto de tuplas de las relaciones de recubrimiento de una subclase sea una extensión de las tuplas de su clase madre. En términos más formales, si la tupla (D_i, S_j) pertenece a C^+ (C^-), la tupla (D_i, S_k) ha de pertenecer a C^+ (C^-) para todo S_k hijo de S_j .

Como vemos, la presencia de la jerarquía simplifica la generación y prueba de candidatos. Proporciona, además, una estrategia sencilla y muy efectiva de obtención de observaciones: solicitar aquellas observaciones que permiten refinar los candidatos consistentes de más alto nivel. Suponer, por ejemplo, que el patrón inicial de observaciones es $\langle 1, 0, 1, ?, ?, ?, ? \rangle$. Este patrón permite rechazar S_2 y sus sucesores, de modo que no es necesario obtener las observaciones de D_4 y D_5 para continuar el diagnóstico. Bastaría con observar D_6 y D_7 para llegar a la solución.

Por último, hay que indicar que, si se imponen condiciones más restrictivas sobre la jerarquía de refinamiento, son posibles otros métodos de razonamiento más poderosos que la eliminación jerárquica. Por ejemplo, si se exige que la clase padre sólo denote la unión de sus clases hijas, es posible razonar por eliminación: la evidencia negativa para una clase se utiliza como evidencia positiva para otra. Como la jerarquía de la Figura 12.11 cumple esta condición adicional, la observación del patrón $\langle 1, ?, 0, ?, ?, ?, ? \rangle$ no sólo permite eliminar S_3 , sino que indica que las únicas soluciones que pueden ser consistentes son S_2 y sus sucesores. Para simplificar la exposición, no explotaremos estas posibilidades y nos limitaremos a utilizar la eliminación jerárquica.

12.4.2.3 Clasificación jerárquica mediante generación guiada por datos

Como en el caso de la clasificación simple, es posible utilizar distintos métodos para realizar la diagnosis sobre un modelo de clasificación jerárquica por recubrimiento. El método que se presenta es una extensión de la clasificación simple guiada por datos, diferenciándose de aquella en el uso de las jerarquías de datos y soluciones y, más importante, utilizando la jerarquía de soluciones para generar una estrategia de obtención de observaciones.

Este método se describe en el Algoritmo 12.3. Esencialmente, utiliza las observaciones iniciales proporcionadas por la subtarea de monitorización para generar un conjunto inicial de hipótesis abstractas. Para refinar estas hipótesis, el método busca observaciones adicionales. Estas observaciones adicionales pueden, a su vez, generar nuevas hipótesis abstractas, que volverán a ser refinadas. Esta forma de intercalar la generación y discriminación de hipótesis con la obtención de observaciones es habitual en muchos dominios de aplicación, donde a partir de unos síntomas iniciales se generan hipótesis cuya confirmación exige recoger datos adicionales. A diferencia de los métodos anteriores, el conjunto de observaciones utilizadas depende de las observaciones iniciales, y el método sólo considera candidatos relacionados con las observaciones que utiliza.

El método se basa en las siguientes suposiciones:

- Se dispone de un procedimiento, *Monitor*, que obtiene el valor de un conjunto reducido de observaciones y nos devuelve en $OBS_{anómala}$ s las observaciones para las que existen candidatos consistentes. Este conjunto de candidatos es mucho menor que el conjunto de posibles soluciones.
- Las soluciones se organizan en una jerarquía de refinamiento que divide las clases de soluciones.
- En cada nivel del espacio de soluciones existe un conjunto de datos que permite discriminar entre ellas. Se dispone de un procedimiento, *Obtener*, que puede solicitar las observaciones correspondientes.
- El espacio de datos se organiza jerárquicamente, facilitando la abstracción de datos. Se dispone de un procedimiento, *Abstraer*, que abstrae los datos según dicha jerarquía.
- Se dispone de un procedimiento, *GenerarNuevosCandidatos*, capaz de obtener de modo eficiente los candidatos más abstractos, consistentes con las observaciones, que aún no hayan sido generados.
- Se utiliza la hipótesis de fallo único.

Una vez obtenidas $OBS_{anómala}$ s por la subtarea de monitorización, el procedimiento se articula en dos bucles iterativos. El más externo genera las soluciones más abstractas y las prueba invocando el procedimiento *ProbarDiscriminar*, buscando observaciones adicionales si es necesario. El bucle interno refina las soluciones hasta llegar a los nodos terminales de la jerarquía, que suelen ser fallos específicos del sistema objeto de diagnóstico. El procedimiento no es determinista, pues no especifica cómo se elige la hipótesis que se quiere refinar, *Candidata_a_refinar*. Aquí son posibles varias estrategias, como el tipo de búsqueda, la asignación de prioridades o el uso de alguna medida de certeza para refinar el candidato más verosímil. Para discriminar entre los hijos de la hipótesis seleccionada, se invoca el procedimiento *ProbarDiscriminar*, que solicita, si es preciso, nuevas observaciones que se van acumulando a las ya obtenidas. Cuando no hay más hipótesis que refinar, el bucle externo comprueba si

las observaciones que se han obtenido en el proceso de refinamiento permiten generar nuevas hipótesis abstractas, en cuyo caso se repite el proceso. Dejamos al lector, en la sección de problemas, el ejercicio de trazar la actividad del sistema para un conjunto dado de observaciones.

Algoritmo 12.3 Diagnosis mediante clasificación jerárquica guiada por datos.

```

Soluciones ← ∅
OBSanómalias ← Monitor()
OBS ← Abstraer(OBSanómalias)
si OBS ≠ ∅ entonces
    mientras se puedan generar nuevos candidatos raíz hacer
        Candidatos ← GenerarNuevosCandidatosRaiz(OBS)
        Soluciones ← ProbarDiscriminar(Candidatos) ∪ Soluciones
        mientras sea posible refinar candidatos hacer
            Soluciones ← Soluciones - {Candidato a refinar}
            Candidatos ← hijos de Candidato a refinar en la jerarquía de soluciones
            Soluciones ← ProbarDiscriminar(Candidatos) ∪ Soluciones
        fin mientras
    fin mientras
fin si
devolver (Soluciones)

ProbarDiscriminar(Candidatos)
NuevasSoluciones ← ∅
obtener OBSadicionales, observaciones necesarias para discriminar entre las hipótesis de Candidatos
NuevasObservaciones ← Abstraer(OBSadicionales)
OBS ← Nuevas observaciones ∪ OBS
para cada Candidato ∈ Candidatos hacer
    si Prueba(Candidato) entonces
        NuevasSoluciones ← NuevasSoluciones ∪ Candidato
    fin si
fin para
devolver (NuevasSoluciones)

```

La Figura 12.12 muestra la particularización del esquema genérico de sistema de diagnosis para la clasificación jerárquica guiada por datos. Vemos que:

- El modelo del sistema es el modelo de clasificación por recubrimiento, $\langle D, S, C^+, C^-, OBS \rangle$, junto a la jerarquía de abstracción de datos y la jerarquía de refinamiento de las soluciones. El modelo no representa de forma explícita ni la estructura ni el comportamiento del mismo. Sólo se dispone de forma explícita de las relaciones entre síntomas y fallos, dados por C^+ y C^- .
- El espacio de datos es D y se solapa con la jerarquía de abstracción de datos del modelo del sistema.
- El espacio de hipótesis es S y se solapa con la jerarquía de refinamiento de soluciones del modelo del sistema.
- No se utiliza el espacio de reparaciones.

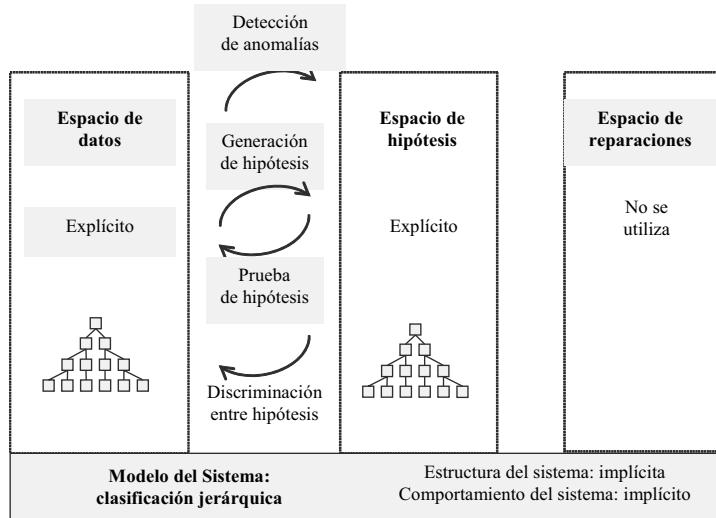


Figura 12.12: Esquema particularizado para la diagnosis mediante clasificación jerárquica.

En relación con las operaciones básicas:

- La subtarea de monitorización es explícita, encargándose de detectar las anomalías del sistema.
- La operación de generar hipótesis obtiene los candidatos más abstractos que son consistentes con las observaciones actuales. Para ello ha de recurrir al modelo del sistema.
- La operación de prueba de hipótesis, soportada por el procedimiento *prueba*, se limita a utilizar las relaciones de recubrimiento del modelo de clasificación para eliminar los candidatos inconsistentes con las observaciones.
- La operación de discriminación de hipótesis es realizado por el procedimiento *ProbarDiscriminar*, que solicita las observaciones necesarias para discriminar entre las hipótesis del mismo nivel, utilizando a continuación el procedimiento *prueba*.

12.4.3 Otros modelos de clasificación simbólica

Los métodos de clasificación simbólica presentados se basan en un modelo de clasificación por recubrimiento muy simple que puede ser ampliado de distintas formas para adaptarlo a problemas reales:

- Ampliar el dominio de valores de las observaciones, para adecuarlos a un área de aplicación específica.

- Añadir medidas de certeza a las observaciones, relaciones de recubrimiento y soluciones, junto a mecanismos de propagación y actualización de la certeza.
- El modelo de recubrimiento es esencialmente conjuntivo: para que una solución sea una explicación es preciso que sea consistente con todas sus observaciones relevantes. Es frecuente ampliarlo con la disyunción.

En ocasiones puede interesar introducir espacios intermedios entre el espacio de datos y el espacio de soluciones, generando **modelos de clasificación multietapa**. En el dominio de la medicina, por ejemplo, algunos sistemas introducen un espacio de “estados patológicos” entre el espacio de datos y el espacio de soluciones.

El modelo de clasificación jerárquico puede ampliarse fácilmente a un modelo de clasificación multietapa. La Figura 12.13 muestra el esquema básico de la clasificación jerárquica, haciendo abstracción de las estructuras de las jerarquías y del modelo de recubrimiento –habitualmente descrito como un proceso de *Equiparación* o emparejamiento, *matching*–, y su versión multietapa.

Finalmente, decir que el modelo de clasificación jerárquica es particularmente útil en combinación con la hipótesis de fallo único: el número de posibles soluciones puede ser manejable y se pueden caracterizar de forma independiente. Aunque puede utilizarse si se considera un número reducido de fallos múltiples, no es viable si hay que considerar cualquier combinación de fallos. La primera razón es el crecimiento exponencial del número de posibles soluciones. La segunda es que la presencia de interacciones entre los síntomas puede requerir caracterizar cada solución compuesta por separado, al no bastar la mera agregación de los síntomas individuales para describir los fallos múltiples. Todo ello complica extraordinariamente el modelo de recubrimiento.

Para abordar este tipo de problemas, manteniéndonos en el ámbito de la clasificación simbólica, son más apropiadas las redes causales. Las redes causales son grafos que proporcionan estructura adicional para reflejar las relaciones causa-efecto entre síntomas y causas. Las redes causales modelan, en términos cualitativos, el comportamiento del sistema en presencia de anomalías. Suelen introducir nodos intermedios para representar estados anómalos del sistema, a partir de los cuales se obtiene el diagnóstico; esta representación más detallada permite un repertorio más amplio de métodos de generación, prueba y discriminación de hipótesis, haciéndolos más adecuados para considerar fallos múltiples.

12.4.4 Implementación de un modelo de clasificación simbólica

Los primeros sistemas expertos que realizaban tareas de diagnosis mediante métodos de clasificación simbólica solían implementarse mediante reglas de un sistema de producción, sin formular de forma explícita el modelo de recubrimiento. Esto generó la idea errónea de que el desarrollo de este tipo de sistemas se limitaba a extraer de los expertos del dominio el conocimiento necesario para realizar la diagnosis, codificarlo en un conjunto de reglas y dejar al intérprete de reglas la estrategia de solución.

En la actualidad, un principio bien asentado es que el conocimiento hay que modelarlo con independencia de la implementación, en el denominado **nivel del**

conocimiento. Las cuestiones relativas al lenguaje de representación en el que se codifica el conocimiento pertenecen al **nivel simbólico**. En este capítulo se ha presentado la clasificación simbólica al nivel del conocimiento, obviando los aspectos relativos a su adquisición e implementación. El conocimiento sobre el dominio está descrito en los modelos de clasificación por recubrimiento y en las jerarquías de abstracción y refinamiento. El conocimiento sobre la estrategia de solución queda especificado en los métodos propuestos.

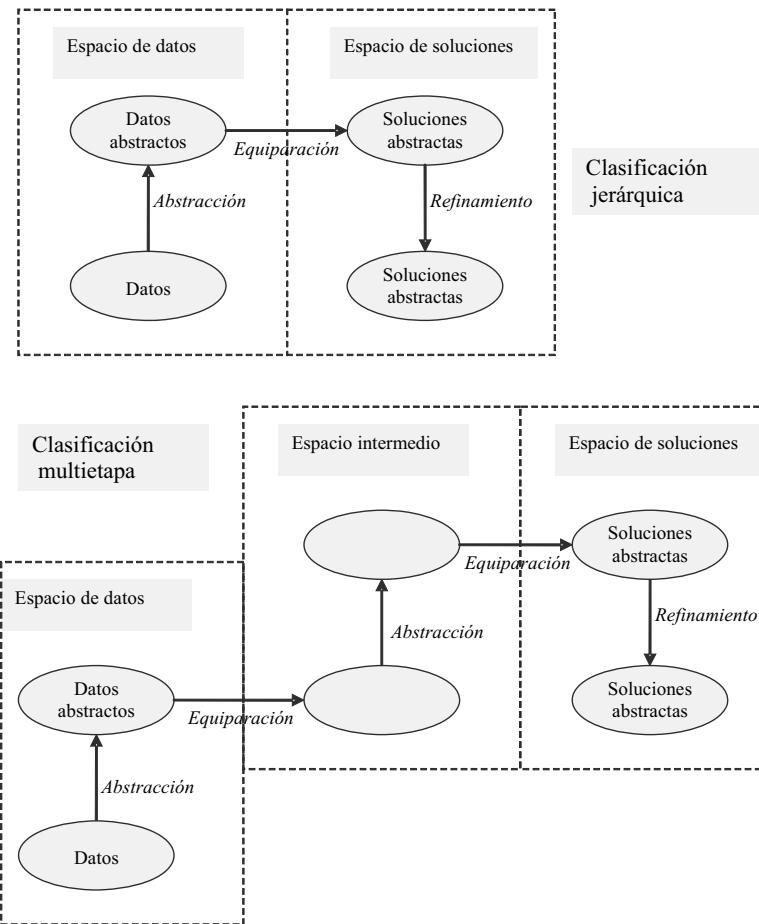


Figura 12.13: Esquema de clasificación jerárquica y de su ampliación a clasificación multietapa.

El diseño y la implementación pueden basarse en cualquier estructura y proceso de cómputo, con tal de que respeten las especificaciones elaboradas en el nivel del conocimiento.

Los sistemas de producción siguen siendo una buena opción para la implementación de un sistema de diagnosis mediante clasificación jerárquica, pues el modelo de recubrimiento admite una traducción casi inmediata a un conjunto de reglas, lo que facilita su codificación inicial y el posterior mantenimiento. Si el motor de inferencias es suficientemente flexible para generar la estrategia descrita por el método de solución, un sistema de producción simplifica enormemente el diseño, la implementación y el mantenimiento del sistema. En la mayor parte de los sistemas de desarrollo, como *CLIPS* o *G2[©]*, esta flexibilidad se consigue integrando el intérprete de reglas con otros paradigmas de programación, de modo que se pueden invocar selectivamente grupos de reglas para realizar distintas operaciones de diagnosis, según una estrategia que se codifica de modo procedimental.

Aun cuando el sistema de producción no permita generar la estrategia deseada, sigue siendo útil durante las etapas de análisis para desarrollar y probar parcialmente el modelo de recubrimiento. Un sistema sencillo permite codificar partes del modelo de recubrimiento, incluidas las jerarquías, y probar su comportamiento con distintos patrones de observaciones. La Figura 12.14 ilustra el proceso de codificación.

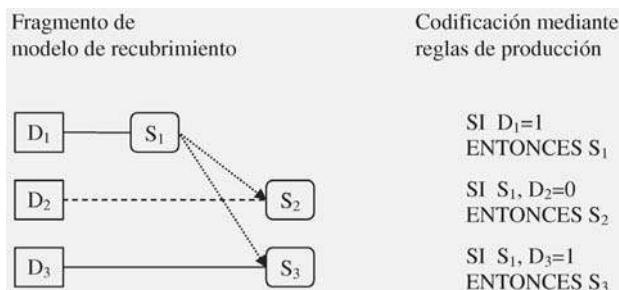


Figura 12.14: Ejemplo de codificación de un fragmento de modelo de recubrimiento.

12.4.5 Un sistema de diagnosis clásico: MYCIN

MYCIN es un sistema de diagnosis médica para infecciones bacterianas y recomendación de terapias. Su importancia histórica radica en que se convirtió en el paradigma de sistema experto de primera generación. Su importancia actual se debe a los exhaustivos análisis al nivel del conocimiento que se han realizado sobre él, no a sus principios de implementación.

Como suele ocurrir en muchos problemas reales, MYCIN combina tareas de distinta naturaleza para alcanzar la solución. Las tareas básicas que realiza MYCIN son:

- Diagnosis. Conceptualmente, MYCIN realiza la tarea de diagnosis mediante clasificación jerárquica. El espacio de datos contiene datos del paciente y abstracciones de los mismos. El espacio de soluciones, clases de enfermedades y enfermedades concretas.

- Terapia. En MYCIN, una terapia es un tratamiento farmacológico. Conceptualmente, MYCIN obtiene la terapia mediante un método de configuración, que selecciona un grupo de fármacos a partir de un conjunto predefinido. La terapia debe considerar aspectos como la utilidad de los fármacos individuales, el historial del paciente, las contraindicaciones, interacciones entre los fármacos o el número total de fármacos administrados. El modelo de clasificación jerárquica no es útil para considerar todos estos aspectos.

Ambas tareas se implementan mediante un sistema de producción basado en reglas, con encadenamiento hacia atrás. MYCIN también emplea otras estructuras para organizar el conocimiento, como el árbol de contextos que asocia a cada paciente y que en la actualidad se diseñaría mediante un sistema de frames. El árbol de contextos mejora la eficiencia del sistema producción, pues cada grupo de reglas se aplica sobre un tipo de nodos del árbol de contextos.

Limitándonos a la tarea de diagnosis, MYCIN utiliza un modelo de recubrimiento jerárquico. El método de clasificación, aunque similar a la generación guiada por datos, presenta algunas variantes. Al igual que este, MYCIN parte de los síntomas (observaciones anómalas) que proporciona el paciente y obtiene las clases abstractas. Durante el proceso de refinamiento también solicita datos adicionales para identificar las infecciones concretas. La diferencia radica en que MYCIN explora todas las clases abstractas, pues busca todas las infecciones posibles.

MYCIN amplía el modelo de recubrimiento cualificando las observaciones y las soluciones mediante factores de certeza, desarrollados específicamente para su uso en este sistema. Aunque desde el punto de vista teórico presenta algunas limitaciones, es un método sencillo muy fácil de incorporar a los sistemas basados en reglas. Más importante que el método de gestión de incertidumbre utilizado por MYCIN, es reconocer el hecho de que en muchas aplicaciones reales hay que enfrentarse a la gestión de la incertidumbre, que queda fuera del alcance de este capítulo.

La Figura 12.15 muestra una regla de MYCIN. Su análisis detallado es de interés para mostrarnos el conocimiento que usa MYCIN y los problemas resultantes de mezclarlo con aspectos estratégicos que no están explícitamente representados. Por ello vamos a proceder a examinar en detalle los elementos que forman la regla.

SI	(1) se dispone de un análisis de sangre, (2) el recuento leucocitario es menor que 2,500
ENTONCES (3) las siguientes bacterias podrían estar causando la infección: E. Coli (.75), Pseudomonas-aeuroginosis (.5), Klebsiella-pneumoniae (.5).	

Figura 12.15: Ejemplo de regla de MYCIN.

La primera cláusula, (1), está relacionada con la estrategia de recogida de datos y está fuera de alcance del modelo de recubrimiento. Simplemente impide que el intérprete evalúe la regla si no se dispone de un análisis de sangre, parte del cual es el recuento leucocitario. Se evita así que el sistema solicite el recuento leucocitario si éste no está disponible. Este tipo de codificación tiene el inconveniente de no informar al intérprete de que la primera cláusula está relacionada con la captura de datos, impiéndole al intérprete reconsiderarla en algún proceso deliberativo sobre la adquisición de nuevos datos. En general, es deseable que el conocimiento relativo a la adquisición de datos esté representado de forma explícita. La solución de MYCIN era aceptable para el estado de la tecnología cuando fue desarrollada.

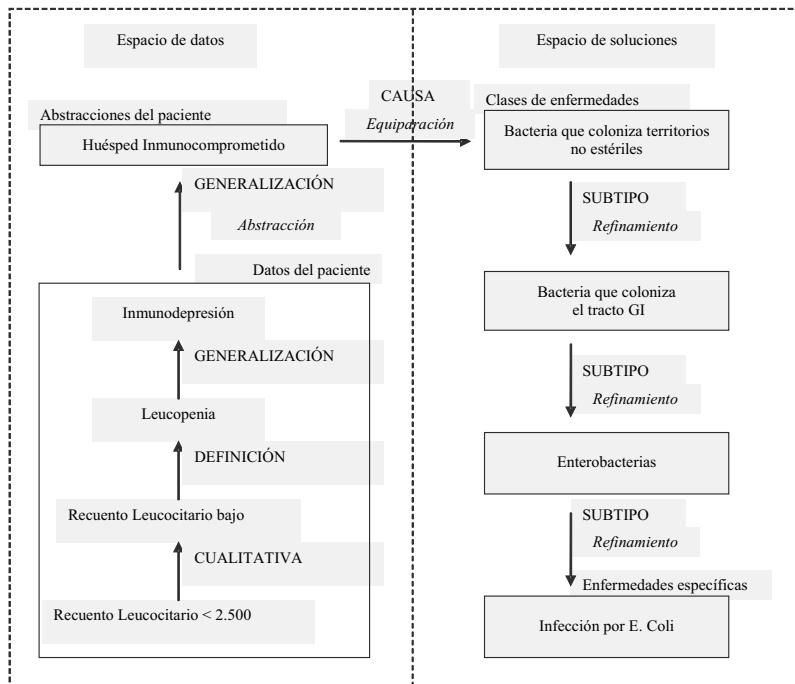


Figura 12.16: Estructura de inferencias de MYCIN.

El resto de la regla es responsable del proceso de clasificación. Tal y como está escrita, asocia directamente una dato de laboratorio con tres tipos de bacterias, asignando un factor de certidumbre a cada uno, que refleja el distinto grado de confianza en cada una de las conclusiones. Para entender la asociación entre el dato y las conclusiones, es necesario hacer explícita la secuencia de pasos elementales de razonamiento que no incluye la regla. Dicha secuencia está descrita en la Figura 12.16, adaptada de Clancey, [Clancey, 1985]. La secuencia de abstracción de datos es la siguiente: si el recuento leucocitario es inferior a 2500, se considera que es bajo y que el paciente padece leucopenia; la leucopenia es un tipo de inmunodepresión; a su vez, los pacientes con inmunodepresión son un tipo de huésped immunocomprometidos. El siguiente paso de

la secuencia asocia el espacio de datos con el espacio de soluciones: cuando un paciente es un huésped immunocomprometido la infección suele estar causada por bacterias que se hallan en territorios no estériles del cuerpo. En la nomenclatura de MYCIN, este paso se denomina asociación heurística. En el modelo de recubrimiento, este paso se reflejaría en una tupla de la relación C^+ . El resto de la secuencia refleja la etapa de refinamiento: el origen más probable de la infección es el tracto gastrointestinal, en el que se encuentran varios tipos de bacterias; en el caso de huéspedes inmunocomprometidos, el elemento patógeno más importante a considerar son las enterobacterias; de ellas, las que con más probabilidad invaden el cuerpo humano a partir del tracto intestinal son *E. coli*, *Pseudomonas* y *Klebsiella*. La secuencia de pasos o inferencias elementales que hemos descrito se corresponde perfectamente con el esquema de la clasificación jerárquica: abstracción de datos, modelo de recubrimiento para alcanzar una solución abstracta y refinamiento de la misma hasta una infección concreta.

MYCIN no representa explícitamente el modelo de recubrimiento mediante reglas: éstas establecen directamente una asociación entre los datos básicos y las infecciones. En la actualidad, la Ingeniería de Conocimiento moderna aboga por identificar el conocimiento necesario para resolver un problema en la etapa de análisis, elaborando modelos de conocimiento. Estos modelos se deben de mantener en la etapa de diseño y sus contenidos reflejarse fielmente en las bases de conocimiento. Los modelos de clasificación presentados en este capítulo son esquemas de modelos de conocimiento de utilidad para la diagnosis.

12.5 Diagnosis basada en modelos: la aproximación basada en consistencia

12.5.1 Motivación

Los métodos de clasificación examinados hasta ahora, diagnosis mediante árboles de fallos y diagnosis mediante modelos de clasificación simbólica, son métodos consolidados que han permitido el desarrollo de numerosas aplicaciones. Estos métodos son adecuados cuando existe un cuerpo de experiencia que permite el desarrollo de los modelos necesarios, si no se dispone de otro tipo de conocimiento y el sistema a diagnosticar permanece estable.

Sin embargo, presentan algunas limitaciones, que se pueden resumir en:

- Relacionadas con el origen del modelo: **Dependencia del dispositivo**. Los modelos del sistema a diagnosticar se elaboran a partir de la experiencia acumulada sobre el comportamiento del sistema en condiciones de funcionamiento anómalo. No se conoce la influencia de los distintos componentes del sistema en el modelo resultante y, en consecuencia, pequeños cambios en el sistema a diagnosticar pueden hacer inservible el modelo.
- Relacionadas con el método de clasificación. Dificultades para diagnosticar **fallos múltiples** e imposibilidad de diagnosticar **fallos no previstos**.

- Relacionadas con el mantenimiento del sistema de diagnosis: dificultades para **mantener las bases de conocimiento** que contienen los modelos. En modelos complejos que requieren adaptaciones y actualizaciones, es difícil garantizar la consistencia de la base de conocimiento.

Los métodos de diagnosis basada en modelos intentan reducir estas limitaciones planteando el problema de forma diferente. En primer lugar, se asume que es posible elaborar un modelo del sistema con capacidad de predicción. La detección de anomalías se realiza comparando el comportamiento observado del sistema con el predicho por el modelo. A continuación, se buscan los elementos del modelo que pueden explicar las discrepancias encontradas. Si se conoce la relación entre los elementos que constituyen el modelo y los componentes del sistema a diagnosticar, sabremos qué componentes del sistema son responsables de la anomalía. La Figura 12.17 resume esta aproximación.

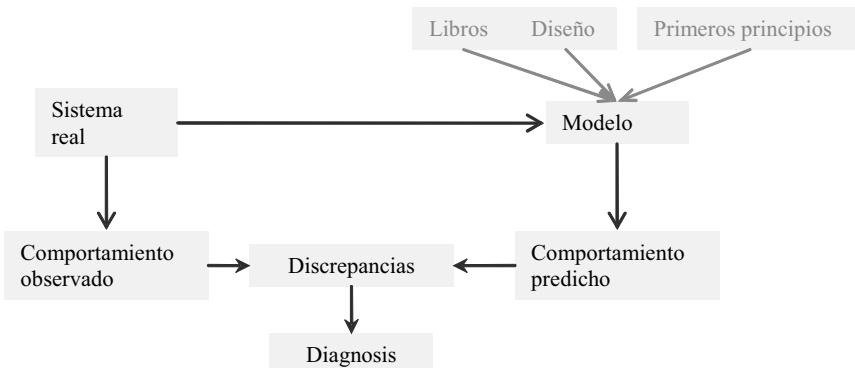


Figura 12.17: Diagnosis basada en modelos: esquema genérico.

El término diagnosis basada en modelos describe una familia de métodos de diagnosis que se basa en las ideas presentadas en el párrafo anterior. En este capítulo nos limitaremos a los métodos desarrollados en el contexto de la IA, denominados genéricamente como **aproximación DX**. La aproximación DX emplea el paradigma de razonamiento basado en modelos, que recurre a manipular los modelos hasta que sus predicciones no generan contradicciones con las observaciones.

La aproximación DX pretende evitar algunas de las limitaciones de los métodos de clasificación, aportando:

- Independencia de la experiencia: se pueden diagnosticar sistemas nuevos si se dispone de sus modelos.
- Independencia del dispositivo: la sustitución de componentes sólo requiere reemplazar sus modelos.
- Capacidad de diagnosticar fallos múltiples y fallos no previstos.

- Solidez y completitud: se pueden diseñar sistemas sólidos y completos, respecto a los modelos utilizados.
- Mantenimiento y reutilización de las bases de conocimiento: las bases de conocimiento se pueden generar a partir de bibliotecas de modelos de componentes.

12.5.2 Diagnosis basada en consistencia

La diagnosis basada en consistencia, DBC, es la formulación principal de la diagnosis basada en modelos en la aproximación DX. Sus características principales son:

- Orientada a componentes: sistemas que se pueden describir como un conjunto finito de componentes interconectados –aunque se ha extendido a sistemas de otro tipo: socioeconómicos, biológicos, procesos, etc.
- Sólo emplea conocimiento de la estructura del dispositivo y del comportamiento de sus componentes.
- Sólo emplea modelos de comportamiento correcto.

Originalmente se formula para sistemas estáticos, por lo que se asume que los valores de las observaciones son constantes y que el comportamiento de los componentes no cambia con el tiempo. Esta suposición excluye los fallos intermitentes.

12.5.2.1 Modelos y predicción del comportamiento

La diagnosis basada en consistencia requiere que el sistema objeto de diagnóstico esté formado por componentes que se unen mediante conexiones entre sus terminales. Los **componentes** suelen ser dispositivos físicos (transistores, resistencias, válvulas, etcétera) que sólo interactúan con el entorno a través de sus terminales. Los **terminales** tienen asociados valores de magnitudes, como voltaje, intensidad o presión. Las **conexiones** se consideran ideales y sólo propagan valores entre terminales. Si fuera necesario, las conexiones reales se pueden modelar como componentes.

La DBC exige que los modelos de los componentes sean **locales** y no realicen ninguna suposición sobre su entorno de trabajo: existencia de otros componentes, estado de otros componentes, etc. Este requisito es necesario para la diagnosis: si los modelos no son locales y utilizan implícitamente suposiciones sobre su entorno, un fallo en dicho entorno invalida el modelo. Además, el uso de modelos locales aporta un beneficio adicional: permite la reutilización de los mismos; el modelo del sistema se puede generar a partir de una biblioteca de modelos de componentes. La exigencia de localidad se traduce en que el modelo de un componente sólo relaciona las variables de entrada y salida de sus terminales y, quizás, los parámetros internos del componente.

Con independencia de la naturaleza del modelo, la DBC explota **todas las restricciones** que este impone sobre las variables de entrada y salida de un componente. No asigna ninguna interpretación causal al modelo y éste se utiliza para predecir el valor de cualquier variable a partir de los datos disponibles.

El uso de modelos locales nos garantiza que la validez de una predicción realizada por el modelo de un componente, o paso de inferencia, sólo depende de una suposición adicional que pocas veces se hace explícita: suposición de funcionamiento correcto del componente que aporta el modelo utilizado para realizar la inferencia. Se denomina **registro de dependencias** al proceso de anotar cuál es la suposición de funcionamiento correcto en que se basa una inferencia. Debido a la localidad de los modelos, en la DBC el registro de dependencias para un paso de inferencia se reduce a registrar el nombre del componente que aporta el modelo junto a las suposiciones de las que quizás dependan los valores que se han utilizado para realizar la predicción.

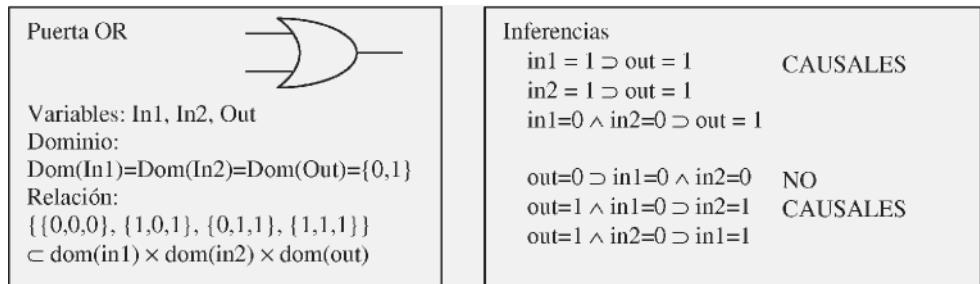


Figura 12.18: Ejemplo: modelo local de una puerta OR e inferencias permitidas.

La Figura 12.18 muestra un componente sencillo, puerta lógica OR, y un modelo relacional de su comportamiento local. Este modelo permite realizar seis inferencias distintas, tres de las cuales no son causales, en el sentido de que no responden al uso que se da al dispositivo. Lo importante es que, si el componente funciona correctamente, las restricciones que impone el modelo han de satisfacerse, con independencia de la causalidad. Así, por ejemplo, si se observa la salida $Out = 0$ y suponemos que la puerta funciona correctamente, se puede inferir $In_1 = 0$ e $In_2 = 0$. Junto a las inferencias, se registraría su dependencia con la puerta OR.

La predicción del comportamiento global del sistema se obtiene a partir de las observaciones, utilizando los modelos locales para predecir las variables no observadas. Cada vez que se realiza una predicción, se registra su dependencia y se propaga su valor a través de las conexiones. La propagación debe realizarse en todas las direcciones posibles, no sólo en la dirección causal.

Ilustraremos este proceso con un ejemplo clásico ya utilizado por Genesereth [Genesereth, 1984] y Davies [Davis, 1982], conocido como sistema multicaja, descrito en la Figura 12.19, un fragmento del cual ha sido utilizado como ejemplo en la sección 12.2. El sistema está formado por tres multiplicadores y dos sumadores, interconectados como indica la figura. Las variables toman valores en el dominio de los números enteros, siendo todas ellas observables, salvo X, Y, Z.

Supongamos que las observaciones toman los siguientes valores: $A = [3]$, $B = [2]$, $C = [2]$, $D = [3]$, $E = [3]$, $F = [10]$, $G = [12]$. A partir de $A = [3]$ y $C = [2]$, utilizando la suposición de funcionamiento correcto de M_1 , se predice $X = 6$. Junto a la predicción, se registra su dependencia: M_1 . De modo similar se predice $Y = 6$ y

se registra su dependencia: M_2 . Finalmente, con $X = 6$ e $Y = 6$ se predice $F = 12$. Como $X = 6$ e $Y = 6$ dependen de M_1 y M_2 respectivamente, $F = 12$ depende de M_1 , M_2 y A_1 . La Figura 12.20 muestra el proceso de propagación y las dependencias de cada dato inferido. La comparación entre el valor predicho $F = 12$ y el valor observado $F = [10]$ indica la presencia de una anomalía, denominada síntoma o discrepancia en el contexto de la DBC. Además, el registro de dependencias proporciona una información importante para la diagnosis: la suposición de que A_1 , M_2 y A_1 funcionan correctamente genera una contradicción con las observaciones; en consecuencia, alguno de estos tres componentes tiene que fallar.

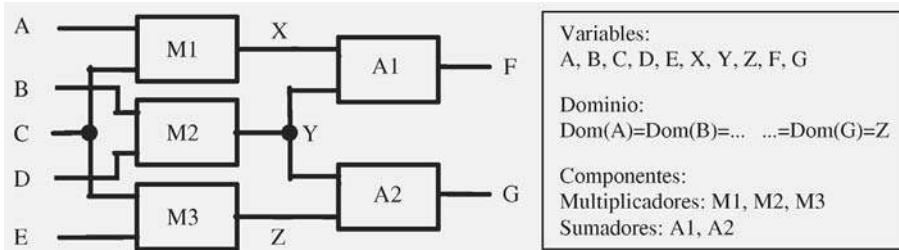


Figura 12.19: Sistema multicaja.

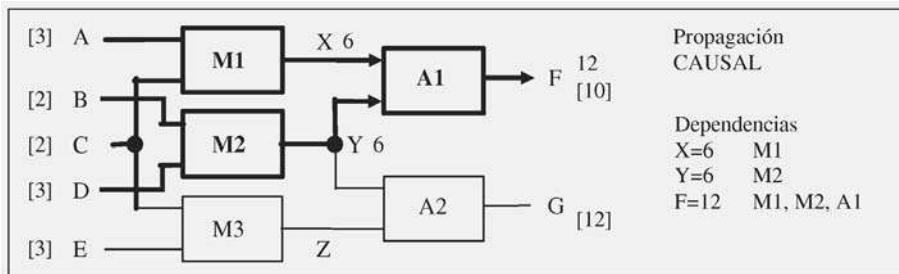


Figura 12.20: Predicción del comportamiento global: propagación causal en el sistema multicaja.

Para detectar todos los síntomas existentes, es preciso explorar todas las posibles predicciones propagándolas por el sistema de todas las formas posibles. La Figura 12.21 muestra un ejemplo de propagación no causal, pues utiliza la observación de la salida de A_1 , $F = [10]$, y el valor predicho de la entrada $X = 6$, para predecir la otra entrada $Y = 4$, que depende de M_2 y A_1 . Como ya hemos obtenido la predicción $Y = 6$ con dependencia M_2 , tenemos un nuevo síntoma (dos valores diferentes para la misma predicción) que depende de M_1 , M_2 y A_1 . Este nuevo síntoma no aporta información adicional al diagnóstico, pues sus dependencias son las mismas que habría para el síntoma detectado con F . Sin embargo, propagando la predicción $Y = 4$ a través del sumador A_2 , se genera la predicción $G = 10$, con dependencias M_1 , M_3 , A_1 y A_2 . Este camino de propagación muestra un nuevo síntoma con otro conjunto de dependencias.

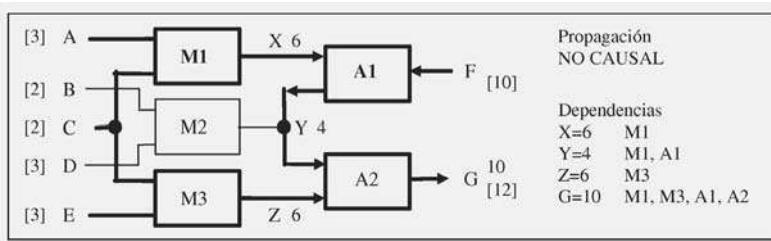


Figura 12.21: Predicción del comportamiento global: propagación no causal en el sistema multicaja.

12.5.3 Modelo formal de la diagnosis basada en consistencia

Reiter, [Reiter, 1987], formalizó teóricamente la DBC describiendo con precisión qué es un problema de diagnosis y cuál es su conjunto de soluciones. Esta formalización utiliza el lenguaje de la lógica de primer orden y sigue siendo el marco conceptual de la DBC. A continuación, se presentan las definiciones y resultados básicos de esta formalización.

Definición 12.1. *Un sistema es un triplete $(DS, COMPS, OBS)$ donde:*

- DS , la descripción del sistema, es un conjunto finito de sentencias de primer orden.
- $COMPS$, los componentes del sistema, es un conjunto finito de constantes.
- OBS , las observaciones del sistema, es un conjunto finito de sentencias de primer orden.

DS define la estructura del sistema y los modelos de funcionamiento correcto de sus componentes. Las conexiones se modelan mediante relaciones de igualdad sobre las señales de los terminales que unen. Para hacer explícita la suposición de funcionamiento correcto, se emplea el literal $\neg AB(x)$, donde el predicado $AB(x)$ representa que el componente x falla. Así, el modelo de un sumador con entradas in_1 e in_2 y salida out se representa:

$$SUM(x) \supset [\neg AB(x) \supset out(x) = in_1(x) + in_2(x)]$$

La definición formal del sistema multicaja es:

- $SD : \{MULT(M_1), MULT(M_2), MULT(M_3), ADD(A_1), ADD(A_2), in_2(M_1) = in_1(M_3), out(M_1) = in_1(A_1), out(M_2) = in_2(A_1), out(M_2) = in_1(A_2), out(M_3) = in_2(A_2),$
 $MULT(x) \supset [\neg AB(x) \supset out(x) = in_1(x) * in_2(x)],$
 $ADD(x) \supset [\neg AB(x) \supset out(x) = in_1(x) + in_2(x)]\}$
- $COMPS : \{M_1, M_2, M_3, A_1, A_2\}$

- $OBS = \{in_1(M_1) = 3, in_2(M_1) = 2, in_1(M_2) = 2, in_2(M_2) = 3, in_2(M_3) = 3, out(A_1) = 10, out(A_2) = 12\}$

Definición 12.2. Dados dos conjuntos de componentes, C_p y C_n , se define $D(C_p, C_n)$ como la conjunción: $\left[\bigwedge_{c \in C_p} AB(c)\right] \wedge \left[\bigwedge_{c \in C_n} \neg AB(c)\right]$.

Definición 12.3. Sea $\Delta \subseteq COMPS$. $D(\Delta, COMPS - \Delta)$ es una diagnosis para $(SD, COMPS, OBS)$ si y sólo si $SD \cup OBS \cup \{D(\Delta, COMPS - \Delta)\}$ es consistente.

Vemos, pues, que la diagnosis se define como una asignación de fallo para los componentes de Δ y una asignación de funcionamiento correcto para los componentes de $COMPS - \Delta$. Como Δ especifica totalmente esta asignación, es habitual denominar candidato a cada conjunto Δ para el cual $D(\Delta, COMPS - \Delta)$ es una diagnosis.

Como se desprende de la definición de diagnosis, el fallo de un componente se interpreta como una desviación de su comportamiento respecto al descrito por su modelo de funcionamiento correcto: si a las suposiciones de funcionamiento correcto de $COMPS - \Delta$ se añade la suposición de funcionamiento correcto de un elemento de Δ , se produce una inconsistencia. En particular, la asignación de fallos que realiza un candidato evita la presencia de cualquier síntoma, ya que la asignación de dos valores distintos a la señal de un terminal genera una inconsistencia lógica.

Si denotamos por n el número de componentes, se pueden generar 2^n conjuntos de candidatos diferentes. Este espacio de candidatos se puede representar de modo eficiente mediante candidatos minimales, gracias a la estructura de retículo (véase la Figura 12.22), que la relación de inclusión induce en este espacio. Por ello, es importante representar el conjunto de diagnosis por las diagnosis minimales.

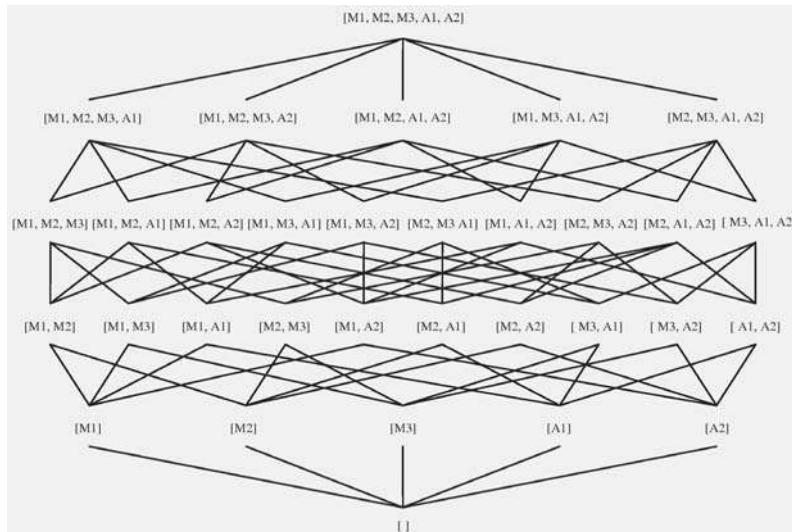


Figura 12.22: Estructura de retículo del espacio de candidatos.

Definición 12.4. Una diagnosis $D(\Delta, COMPS - \Delta)$ es una diagnosis minimal si y sólo si no existe un subconjunto propio de Δ, Δ' , tal que $D(\Delta', COMPS - \Delta')$ es una diagnosis.

La siguiente proposición, consecuencia de la definición de diagnosis, recoge la intuición de que, si todos los componentes funcionan correctamente, no debe observarse un comportamiento anómalo.

Proposición 12.1. $D(\emptyset, COMPS - \emptyset)$ es una diagnosis para $(SD, COMPS, OBS)$ si y sólo si $SD \cup OBS \cup \{\neg AB(c) / c \in COMPS\}$ es consistente. Además, es la única diagnosis minimal.

Las siguientes definiciones introducen el concepto de conflicto, esencial en la DBC.

Definición 12.5. Un literal-AB es $AB(c)$ o $\neg AB(c)$ para algún $c \in COMPS$.

Definición 12.6. Una cláusula-AB es una disyunción de literales-AB que no contiene literales complementarios.

Definición 12.7. Sea C una cláusula-AB. C es un conflicto para $(SD, COMPS, OBS)$ si y sólo si C es consecuencia lógica de $SD \cup OBS$.

Para simplificar la notación, los conflictos suelen representarse mediante el conjunto de componentes referenciados por los literales-AB del conflicto. Para evitar confusiones, los candidatos se suelen representar entre corchetes, $[]$.

Definición 12.8. Un conflicto C es un conflicto minimal si y sólo si no existe un subconjunto propio de C , C' , tal que C' es un conflicto.

A partir de la definición formal del sistema multicaja, se puede demostrar que se generan dos conflictos minimales:

$$\begin{aligned} C_1 : \quad & AB(M_1) \vee AB(M_2) \vee AB(A_1) - \{M_1, M_2, A_1\} - \\ C_2 : \quad & AB(M_1) \vee AB(M_3) \vee AB(A_1) \vee AB(A_2) - \{M_1, M_3, A_1, A_2\} - \end{aligned}$$

El conflicto C_1 , $\{M_1, M_2, A_1\}$, representa que M_1 o M_2 o A_1 fallan. Precisamente M_1 , M_2 y A_1 son las dependencias de la predicción $F = 12$ que causa un síntoma. El concepto de conflicto formaliza la intuición de que si las suposiciones de funcionamiento correcto generan un síntoma, alguna de estas suposiciones debe ser falsa. Examinando la Figura 12.21 comprobamos que las dependencias asociadas a la predicción $G = 10$, que genera otro síntoma, se corresponden con el conflicto C_2 . Nótese que el concepto de conflicto se define en términos lógicos y, teóricamente, cualquier sistema de inferencia completo nos permitiría derivarlos; la propagación local de todas las inferencias posibles a través de la estructura del dispositivo junto al registro de dependencias es un método de cómputo efectivo, pero no el único. El siguiente teorema es el resultado fundamental de la teoría de DBC:

Teorema 12.1. $D(\Delta, COMPS - \Delta)$ es una diagnosis minimal para $(SD, COMPS, OBS)$ si y sólo si Δ es un conjunto de corte minimal de la familia de conflictos minimales de $(SD, COMPS, OBS)$.

Este teorema proporciona un método sistemático para obtener las diagnosis minimales:

1. Obtener los conflictos minimales.
2. Generar los candidatos minimales como los conjuntos de corte minimal de los conflictos.

La mayor parte de los sistemas de diagnosis basados en consistencia utilizan este método, que se puede aplicar de forma incremental a medida que se encuentran nuevos conflictos minimales, pues los conjuntos de corte se pueden generar de modo incremental.

Ilustraremos el método con el sistema multicaja. A partir de sus dos conflictos minimales, se obtienen los siguientes conjuntos de corte minimal: $\{M_1\}$, $\{A_1\}$, $\{M_2, M_3\}$, $\{M_2, A_2\}$. La Figura 12.23 muestra el proceso. Para cada Δ_i se obtiene una diagnosis minimal, D_i . D_1 y D_2 son hipótesis de fallo único. D_3 y D_4 son hipótesis de fallo doble. D_3 , por ejemplo, indica que el fallo simultáneo de M_2 y M_3 es consistente con la descripción del sistema y las observaciones, no siendo posible un fallo único en M_2 o en M_3 . Desde el punto de vista de la DBC, las cuatro diagnosis tienen el mismo estatus y para discriminarlas hay que inyectar información adicional, bien sea en forma de nuevas observaciones, probabilidad de fallo a priori, etc.

Algoritmo 12.4 Generación de candidatos guiada por conflictos.

```

FamiliaCandidatos ← ∅
para cada Conflicto ∈ ConflictosMinimales hacer
    CandidatosActuales ← FamiliaCandidatos
    para cada Candidato ∈ CandidatosActuales hacer
        si Candidato ∩ Conflicto = ∅ entonces
            FamiliaCandidatos ← FamiliaCandidatos - Candidato
        para cada Componente ∈ Conflicto hacer
            NuevoCandidato ← Candidato ∪ {Componente}
            FamiliaCandidatos ← FamiliaCandidatos ∪ NuevoCandidato
        fin para
        Eliminar elementos duplicados y no minimales de FamiliaCandidatos
    fin si
    fin para
fin para
devolver (FamiliaCandidatos)

```

El Algoritmo 12.4 describe un procedimiento que genera incrementalmente los candidatos a partir de los conflictos. El procedimiento construye un primer conjunto de candidatos con los elementos de un conflicto. Seguidamente, comprueba si estos candidatos contienen elementos de otro conflicto; si no es así, reemplaza el candidato con todos aquellos que se pueden generar añadiendo, uno por uno, cada elemento del

conflicto al candidato eliminado. Se propone como ejercicio para el lector comprobar el funcionamiento del procedimiento sobre el sistema multicaja.

Obsérvese que el efecto de añadir un nuevo conflicto minimal a los ya existentes consiste en podar candidatos minimales. Inicialmente, cuando no se ha considerado ningún conflicto, el único candidato minimal es $[]$. La Figura 12.24 describe la situación tras procesar el primer conflicto: $[]$ se reemplaza por $[M_1]$, $[M_2]$ y $[A_1]$ mientras que $[M_3]$, $[M_2]$ y $[M_3, A_2]$ son eliminados. Al procesar el segundo conflicto, Figura 12.25, se elimina el candidato minimal $[M_2]$, que se reemplaza por $[M_2, M_3]$ y $[M_2, A_2]$.

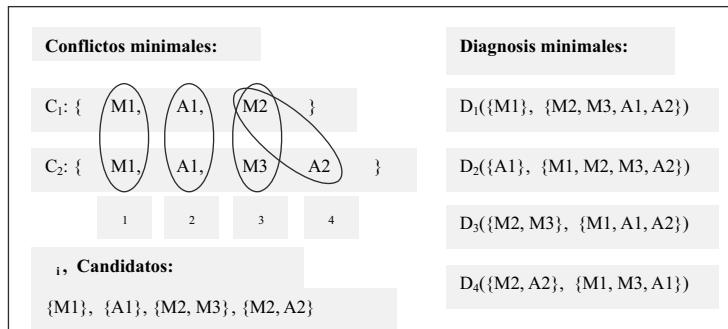


Figura 12.23: Diagnósticos minimales como conjuntos de corte de los conflictos minimales.

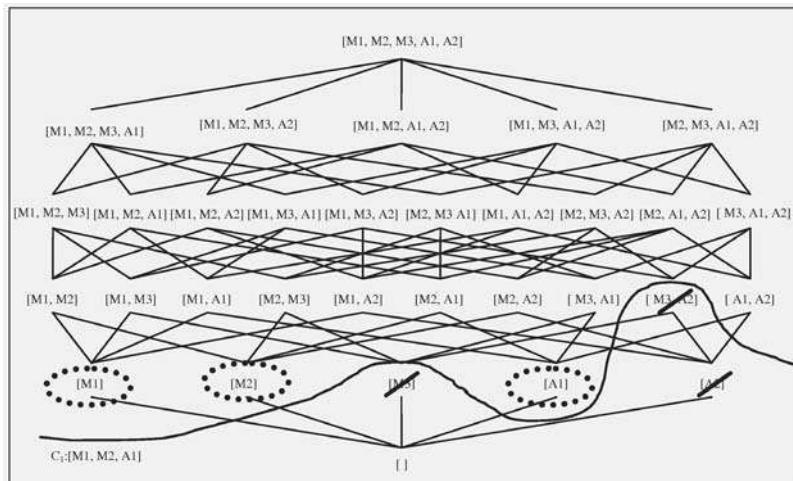


Figura 12.24: Generación incremental de candidatos tras el primer conflicto.

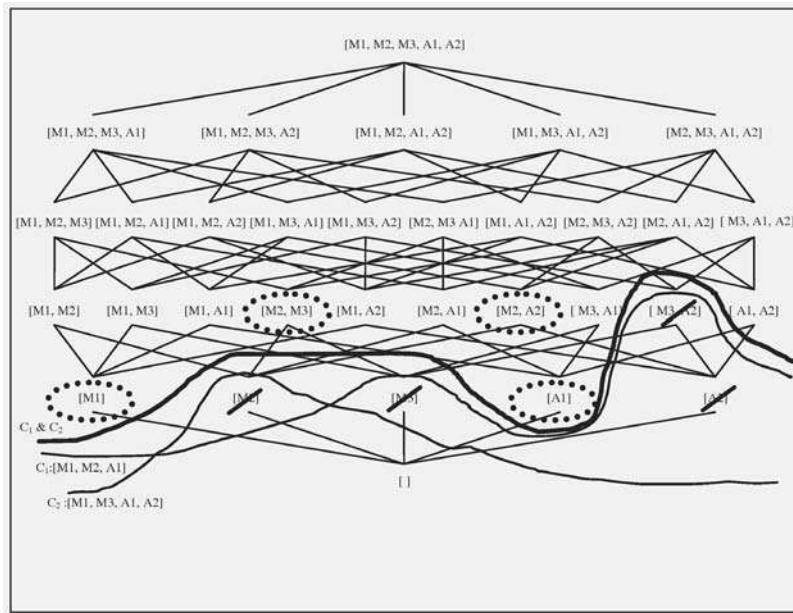


Figura 12.25: Generación incremental de candidatos tras el segundo conflicto.

12.5.4 Limitaciones de la diagnosis basada en consistencia

La DBC establece un marco conceptual sencillo y elegante para abordar problemas de diagnóstico. La propuesta es general e independiente del dominio de aplicación, evitando los problemas asociados al uso de modelos extraídos de la experiencia. El sistema es capaz de localizar fallos múltiples y fallos desconocidos a partir de modelos de comportamiento y estructura del sistema a diagnosticar.

Plantea, sin embargo, otros inconvenientes que dificultan su aplicación a problemas reales:

- Necesidad de modelos locales con capacidad predictiva. Salvo en dominios sencillos, como la electrónica digital, no es fácil obtener este tipo de modelos. En realidad, la mayor dificultad para aplicar la DBC a sistemas reales es el modelado de los mismos.
- La generación de conflictos es un proceso computacionalmente costoso. En sistemas con numerosos componentes, es preciso utilizar alguna heurística para guiar el proceso de generación de conflictos.
- Al utilizar sólo modelos de funcionamiento correcto, no se dispone de información de cómo fallan los componentes. Esto se traduce en:
 - Dificultades para la identificación de fallos. En el marco de la DBC, esto requiere añadir modelos de comportamiento incorrecto, denominados

modelos de fallos. Aparte de aumentar la complejidad computacional, estos modelos son más difíciles de obtener.

- La DBC puede localizar fallos lógicamente posibles que no pueden darse en el mundo físico.

La Figura 12.26 muestra un sistema propuesto por Struss y Dressler, [Struss y Dressler, 1989], para ilustrar los inconvenientes de no usar modos de fallos. El sistema está formado por una pila y tres bombillas conectadas en paralelo. Se observa que B_1 y B_2 no lucen mientras B_3 luce. Propagando en dirección causal, se obtienen los conflictos minimales $\{P, B_1\}$ y $\{P, B_2\}$. Propagando en dirección no causal, se obtienen los conflictos $\{B_3, B_2\}$ y $\{B_3, B_1\}$. Los candidatos minimales son $[B_1, B_2]$ y $[P, B_3]$. El candidato $[B_1, B_2]$ se corresponde con un comportamiento posible del circuito: las bombillas B_1 y B_2 están fundidas mientras que P y B_3 funcionan correctamente. El candidato $[P, B_3]$ no tiene sentido, pues indica que B_1 y B_2 funcionan correctamente, P no tiene carga y B_3 luce sin corriente. Ciertamente, esta asignación de comportamiento a B_3 no es admisible, pues no se puede dar en el mundo real.

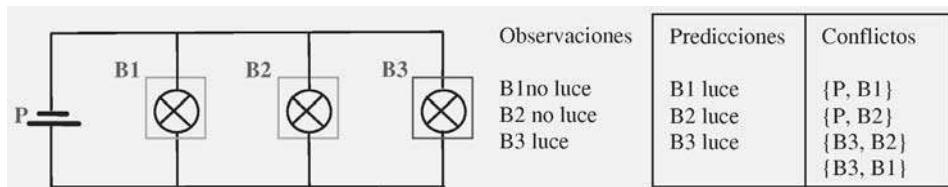


Figura 12.26: Inconvenientes de no utilizar modelos de fallos.

12.5.5 Paradigma computacional: General Diagnostic Engine

General Diagnostic Engine, GDE, fue desarrollado por Kleer y Williams, [de Kleer y Williams, 1987]. GDE introduce, de modo informal, los conceptos básicos que luego formalizó Reiter. La importancia de GDE se debe a:

- Fue el primer sistema de diagnosis basada en modelos con capacidad para diagnosticar fallos múltiples.
- Es el principal paradigma computacional de la DBC y sigue usándose hoy en día.
- Sigue siendo una referencia para la comparación de nuevas propuestas de sistemas de diagnosis basada en modelos en la aproximación DX.

GDE realiza la diagnosis de forma incremental, mediante un proceso iterativo de generación de candidatos y refinamiento de los mismos. Para cada conjunto de observaciones, GDE realiza las siguientes operaciones:

- Detectar todos los síntomas.
- Identificar los conflictos minimales.
- Generar los candidatos minimales.

GDE utiliza la teoría de la información para proponer nuevos puntos de medida que le permitan refinar los candidatos. La Figura 12.27 describe el ciclo de trabajo de GDE.

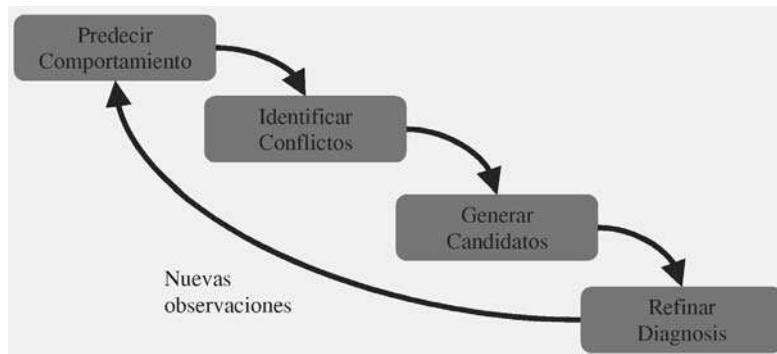


Figura 12.27: Ciclo de diagnosis en GDE.

12.5.5.1 Detección de síntomas

GDE separa totalmente los procedimientos de predicción del comportamiento del método de diagnosis. Para predecir el comportamiento, utiliza un mecanismo de inferencia acoplado con un mecanismo de registro de dependencias que le permite obtener computacionalmente el comportamiento global del sistema y las dependencias de las predicciones, a partir de los modelos locales de los componentes y la estructura de dispositivo. Como mecanismo de registro de dependencias, GDE utiliza un sistema de mantenimiento de la razón basado en suposiciones, ATMS.

El procedimiento de inferencia debe cumplir los requisitos básicos necesarios para trabajar con un mecanismo de registro de dependencias:

- Se puede construir una dependencia para cada paso de inferencia.
- Una predicción se considera cierta si y sólo si las suposiciones de alguna de sus dependencias son ciertas. En el ejemplo del sistema multicaja, la predicción $F = 10$ con dependencia $\{M_1, M_2, A_1\}$ se considera cierta si y sólo si M_1, M_2 y A_1 funcionan correctamente.
- El orden de las inferencias es irrelevante.
- El procedimiento de inferencia es monótono: no se eliminan inferencias ya realizadas.

Estos requisitos son muy generales, de modo que muchos sistemas de inferencia los satisfacen. GDE utiliza como procedimiento de inferencia un mecanismo de propagación de restricciones. Los modelos locales de los componentes se utilizan como restricciones entre las variables de sus terminales. Los valores obtenidos se propagan en todas las direcciones. GDE reproduce el proceso de propagación que hemos mostrado en la sección 12.5.5.1.

12.5.5.2 Conflictos y generación de candidatos

El acoplamiento del sistema de inferencias y el ATMS permite obtener automáticamente los conflictos minimales. Cuando se detecta un síntoma (variables con dos valores distintos), las dependencias registradas por el ATMS son un conflicto. El ATMS sólo mantiene dependencias minimales, de modo que proporciona automáticamente los conflictos minimales.

A partir de los conflictos minimales, GDE obtiene los candidatos minimales de forma incremental. Cada nuevo conflicto minimal encontrado permite eliminar algún candidato. Para obtener todos los conflictos minimales es preciso que el procedimiento de inferencia sea completo, para garantizar que se encuentran todos los síntomas. Por motivos de eficiencia, no siempre es viable utilizar sistemas de inferencia completos. En este caso algunos síntomas no serán detectados, lo que puede hacer que no se encuentren todos los conflictos minimales. La única consecuencia es que se eliminarán menos candidatos que en el caso ideal, pero no se eliminan candidatos válidos.

12.5.5.3 Discriminación de candidatos

Una vez generados los candidatos minimales con las observaciones disponibles, GDE intenta discriminar los candidatos proponiendo puntos de prueba para obtener nuevas observaciones.

GDE asume que la adquisición de medidas no afecta al sistema objeto de diagnosis e intenta seleccionar el punto de prueba que proporcione, por término medio, la máxima información para discriminar entre los candidatos.

GDE emplea una estrategia *one-step look-ahead* evaluando la calidad de un punto de prueba a partir de la entropía de la distribución de probabilidad de los candidatos resultantes para cada posible valor de la observación.

No nos vamos a extender en el método utilizado por GDE para determinar la entropía de un punto de prueba. El lector interesado puede consultar [Stefik, 1995].

12.5.5.4 Comparación con el esquema genérico de sistema de diagnosis

La principal diferencia entre GDE y los restantes sistemas de diagnóstico que hemos examinado en este capítulo es la naturaleza del modelo. Para razonar sobre el sistema, GDE utiliza un modelo que describe la estructura y el comportamiento del sistema a diagnosticar. No se utilizan asociaciones entre hipótesis de fallos y observaciones. El proceso de diagnosis consiste en eliminar las hipótesis de fallo que no son consistentes con las observaciones y el modelo, generando las hipótesis minimales.

GDE no introduce estructura adicional en el espacio de datos. Este sólo contiene observaciones y posibles puntos de prueba. Sin embargo, explota la estructura del espacio de hipótesis que gestiona de forma compacta representando los conflictos y los candidatos por sus conjuntos minimales.

Al igual que los métodos de clasificación simbólica examinados en la sección 12.4, GDE no razona con el espacio de reparaciones. Siendo un sistema orientado a la diagnosis de dispositivos constituidos por componentes interconectados, las acciones de reparación se limitan a reemplazar o reparar los componentes identificados por los candidatos minimales.

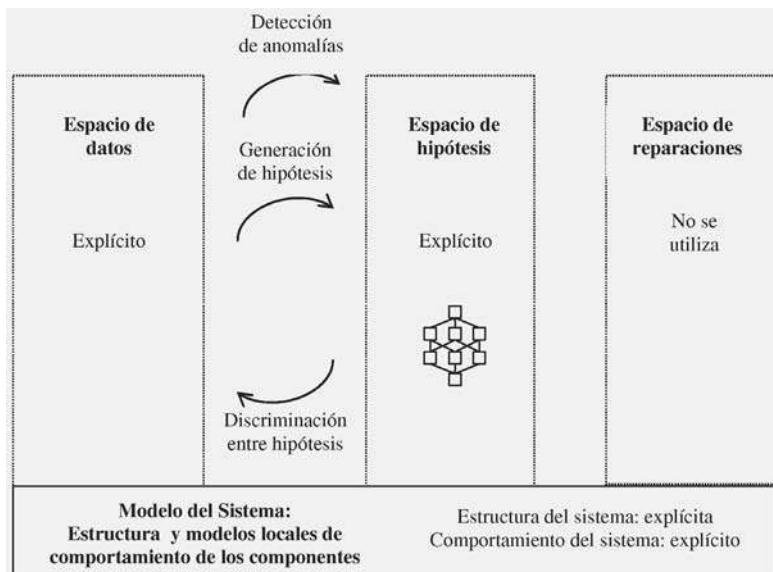


Figura 12.28: Esquema particularizado para GDE.

Respecto a las tareas de diagnosis y sus operaciones básicas, tenemos:

- La tarea de monitorización o detección de anomalías está fuertemente acoplada a la tarea de diagnosis. El propio GDE puede encargarse de la monitorización del sistema. Es posible utilizar otros métodos de monitorización para poner en marcha el proceso de diagnosis, pero es GDE el que busca exhaustivamente las anomalías presentes.
- La generación de hipótesis está totalmente determinada por los conflictos minimales. La búsqueda en el espacio de hipótesis se reduce de forma drástica por el uso de hipótesis minimales. La contrapartida es el esfuerzo de búsqueda para obtener los conflictos, que GDE resuelve mediante un mecanismo de propagación de restricciones acoplado a una ATMS.
- No hay prueba de hipótesis, pues el método de generación de candidatos garantiza que son consistentes con las observaciones actuales.

- El proceso de discriminación de hipótesis solicita una nueva medida en cada ciclo de trabajo. Se selecciona aquella medida que se espera que tenga mayor capacidad de discriminación de candidatos.

GDE ha dado origen a numerosos sistemas de diagnóstico que, de una forma u otra, han intentado incrementar su capacidad de diagnóstico y sus dominios de aplicación. Sherlock, [de Kleer y Williams, 1989], incorpora modelos de funcionamiento incorrecto que permiten identificar fallos y que evitan la generación de candidatos que corresponden a modos de fallos que no se producen en la realidad. XDE, [Hanscher, 1989], incorpora jerarquías de componentes y abstracciones del comportamiento temporal que permiten diagnosticar sistemas dinámicos con numerosos componentes, si bien se limita al dominio de la electrónica digital. Más recientemente se han realizado esfuerzos para extender estas técnicas a sistemas dinámicos continuos que no se describen mediante agregación de componentes. Un ejemplo representativo es TIGER, [Travé-Massuyès y Milne, 1997], que incluye un módulo de diagnosis basada en consistencia para el diagnóstico de fallos en turbinas generadoras.

12.6 Métodos y modelos para la diagnosis

El modelo del sistema objeto de diagnóstico y el método de diagnosis determinan las propiedades básicas de un sistema de diagnóstico. Ambos están relacionados, pues cada método requiere un tipo de modelo del sistema.

	Métodos de clasificación	Métodos basados en casos	Métodos basados en modelos
Modelos implícitos	Diagnosis mediante modelos de clasificación de caja negra: <ul style="list-style-type: none"> • Redes de neuronas • SVM 	Diagnosis basada en casos	
Modelos explícitos	Diagnosis mediante árboles de fallos Diagnosis mediante modelos de clasificación simbólica: <ul style="list-style-type: none"> • Clasificación simple • Clasificación jerárquica • Redes causales 		Diagnosis basada en modelos: DX Orientado a componentes: <ul style="list-style-type: none"> • Modelos estructurales • Modelos funcionales • Modelos de estructura y comportamiento Orientado a procesos <ul style="list-style-type: none"> • Modelos causales • Modelos globales de simulación FDI Modelos globales de simulación

Figura 12.29: Aproximaciones al diagnóstico, según el método de diagnosis y el modelo del sistema a diagnosticar utilizado.

Sin ánimo de exhaustividad, la Figura 12.29 muestra las principales aproximaciones computacionales a la tarea de diagnosis, clasificadas atendiendo al método de diagnosis y al modelo del sistema objeto de diagnóstico.

Vamos a distinguir tres familias de métodos: métodos de clasificación, métodos basados en casos y métodos basados en modelos.

12.6.1 Métodos de clasificación

Los métodos de clasificación abordan el problema de diagnosis como una tarea de clasificación: dado un conjunto finito y predeterminado de posibles causas, utilizan las observaciones para seleccionar un subconjunto de aquellas como responsables de la anomalía. En la medida en que el método de diagnosis se limite a seleccionar elementos de un conjunto predeterminado y de alguna manera codificados en el modelo, se hablará de método de clasificación.

Comenzaremos tipificando los modelos de clasificación como implícitos o explícitos. En los modelos de clasificación implícitos, las relaciones entre las clases y las observaciones que permiten seleccionarlos están representadas de forma que no son comprensibles para el usuario. Los sistemas de diagnosis que usan estos modelos son capaces de detectar, localizar e identificar fallos, pero no pueden proporcionar información adicional sobre el proceso de diagnosis. De manera genérica, denominaremos a estos modelos “modelos de clasificación de caja negra”. Generalmente, estos métodos enfocan el problema de diagnosis como un problema de clasificación de patrones. Entre ellos, los más frecuentes son los clasificadores inducidos mediante técnicas de aprendizaje: redes neuronales, máquinas de vectores soporte, etc. Aunque la inducción de clasificadores para problemas de diagnóstico presenta algunas peculiaridades propias de esta tarea específica, remitimos al lector al módulo 5 de esta misma obra, en el que se presentan distintas técnicas de aprendizaje.

En los modelos de clasificación explícitos, las relaciones entre los síntomas y los fallos están expresamente representadas, de forma que el sistema puede utilizarlas (y reutilizarlas) de distintas formas. En particular, pueden facilitar la generación de explicaciones, aspecto importante en algunos dominios de aplicación. Vamos a distinguir tres tipos de modelos de clasificación explícitos: árboles de fallos, modelos de clasificación simbólica y modelos de clasificación estadística.

Los árboles de fallos, presentados en la sección 12.3, son esencialmente diagramas de flujo que codifican la estrategia a seguir para alcanzar un diagnóstico y/o proponer una acción de reparación. De todos los modelos explícitos es el que menos información proporciona sobre el sistema objeto de diagnosis, pudiendo afirmarse que más que un modelo del sistema es un plan de actuación para realizar la diagnosis.

Los modelos de clasificación simbólica representan, de forma explícita, las relaciones entre síntomas y fallos. Junto con los árboles de fallos, son los modelos que más se han utilizado para el desarrollo de Sistemas Expertos para la diagnosis. Por motivos históricos, a este tipo de sistemas también se les denomina Sistemas Basados en Conocimiento para la diagnosis. Vamos a considerar tres tipos de modelos de clasificación simbólica: clasificación simple, clasificación jerárquica y redes causales. Un modelo de clasificación simple se limita a proporcionar las relaciones de los fallos a sus síntomas. Los modelos de clasificación jerárquica añaden estructura al modelo, generalmente en forma de jerarquía de datos e hipótesis. Ninguno refleja de forma explícita las propiedades estructurales o de comportamiento del sistema objeto de diagnóstico, más allá

de asociar fallos a síntomas. Las redes causales son grafos que proporcionan estructura adicional para reflejar las relaciones causa-efecto entre síntomas y causas. Las redes causales modelan, en términos cualitativos, el comportamiento del sistema en presencia de anomalías. Suelen introducir nodos intermedios para representar estados anómalos del sistema, a partir de los cuales se obtiene el diagnóstico; esta representación más detallada permite un repertorio más amplio de métodos de generación y prueba de hipótesis, haciéndolos más adecuados para considerar múltiples fallos.

Los modelos estadísticos también enfocan el problema de diagnosis como un problema de clasificación de patrones, pero, a diferencia de los modelos implícitos, la estructura de los modelos es conocida y, en muchos casos, los modelos son comprensibles. Aunque su estudio excede los objetivos de este capítulo, creemos interesante mencionarlos por su importancia y por los numerosos sistemas que se han desarrollado con este tipo de técnicas. Los clasificadores bayesianos son un buen ejemplo de este tipo de sistemas. Mención especial merecen las redes bayesianas, modelos gráficos que permiten la propagación y actualización de probabilidades a posteriori al obtener nuevas observaciones. Las redes bayesianas son particularmente populares en aplicaciones de diagnóstico médico. Aunque presentan cierta similitud con las redes causales que hemos incluido entre los clasificadores simbólicos, su naturaleza formal y su fundamento en la teoría de la probabilidad recomiendan diferenciarlas de las primeras. Una primera introducción a este tipo de modelos puede encontrarse en el capítulo 6 de este libro.

12.6.2 Métodos basados en casos

Los métodos basados en casos utilizan el paradigma de razonamiento basado en casos para realizar la diagnosis. Este paradigma no dispone de un modelo explícito del sistema. El razonamiento basado en casos trata de imitar la capacidad humana de resolver nuevos problemas según la experiencia acumulada en la resolución de problemas similares en el pasado. Las experiencias pasadas se codifican en casos, que describen tanto el problema como su solución. Cuando el sistema tiene que resolver un problema, accede a la base de casos para recuperar aquellos que más se parecen, según algún criterio, al problema actual. En la medida de lo posible, el sistema tratará de adaptar las soluciones recuperadas al problema actual. Cuando se resuelve algún problema novedoso, este se almacena en la base de casos para incrementar la experiencia del sistema.

El paradigma de razonamiento basado en casos es particularmente interesante en aquellos dominios en que no es fácil elaborar un modelo explícito del sistema a diagnosticar, pero sin embargo se dispone de experiencia previa que pueda ser fácilmente reutilizada. Ha sido utilizado con éxito en aplicaciones de soporte al cliente, como herramienta de ayuda a los operadores de soporte para focalizar rápidamente el tipo de problema que presenta el usuario. Estos sistemas se conciben como sistemas de ayuda que cooperan con los responsables de realizar el diagnóstico más que como agente autónomos. A pesar de su importancia, tampoco nos extenderemos en este capítulo en esta aproximación, remitiendo al lector interesado al capítulo 22 de este libro, dedicado a este paradigma de razonamiento.

12.6.3 Métodos basados en modelos

Dado que todos los métodos de diagnosis necesitan algún tipo de modelo del sistema para realizar el diagnóstico, esta es una denominación que puede resultar confusa. Con ella se pretende destacar que estos métodos utilizan modelos explícitos, generados con métodos más sistemáticos y con capacidad de predecir el comportamiento del sistema. La tarea de diagnóstico se realiza analizando las diferencias entre el comportamiento predicho por el modelo y el comportamiento observado. Esto contrasta con la mayoría de las aproximaciones anteriores, donde, de forma más o menos explícita, el modelo incluye relaciones entre síntomas y fallos, que son el soporte básico de las operaciones de diagnosis.

Existen dos aproximaciones básicas en esta familia de métodos. Por una parte, y con origen en la IA, está la aproximación DX, que utiliza el paradigma del razonamiento basado en modelos: estas técnicas recurren a manipular directamente los modelos hasta que sus predicciones no generan contradicciones con las observaciones. Como resultado de esta manipulación, se obtiene la diagnosis. Por otra parte, con origen en la teoría de control, está la aproximación FDI (*Fault Detection and Identification*). Esta segunda aproximación se centra más en los problemas de detección de anomalías, utilizando procedimientos de toma de decisión más sencillos para realizar el diagnóstico. A cambio, son más robustas en la detección.

Históricamente, la aproximación DX ha utilizado modelos orientados a componentes, de modo que el comportamiento global del sistema se obtiene por composición del comportamiento de sus elementos constituyentes. Entre este tipo de modelos, los más populares son los modelos de estructura y comportamiento, como los utilizados en la sección 12.5. También se han utilizado modelos orientados a procesos, como los modelos causales o los modelos globales de simulación. A diferencia de las redes causales, mencionadas en la sección 12.6.1 estos modelos causales utilizan nodos para representar variables del sistema y arcos para indicar la relación, muchas veces de forma cualitativa, entre dos variables conectadas por un arco, permitiendo predecir cualitativamente el comportamiento del sistema. Los modelos globales de simulación modelan el comportamiento del sistema mediante un conjunto de ecuaciones que permiten predecir el comportamiento global del mismo. Cuando se utilizan modelos globales de simulación, la aproximación DX recurre a precompilar las dependencias para identificar los conflictos.

La aproximación FDI habitualmente utiliza modelos globales del sistema para generar un vector de residuos, o diferencias entre valores observados y predichos. En la mayoría de los casos, los residuos toman valores binarios, de forma que se tiene un conjunto finito de posibles valores del vector de residuos. Esta familia de técnicas utiliza la hipótesis de fallo único, lo que le permite asociar cada posible valor del vector de residuos con una o más hipótesis de fallos.

12.6.4 Origen de los modelos

En la mayor parte de los problemas de diagnóstico, la principal dificultad se halla en elaborar un modelo del sistema de utilidad para la tarea de diagnosis. En muchos

casos, es la naturaleza de la información o conocimiento disponible lo que sugiere cuál es la mejor aproximación para un problema dado. Para simplificar, consideraremos tres posibles fuentes de conocimiento:

- Históricos de datos de fallos.
- Experiencia humana en la tarea de diagnosis.
- Primeros principios de funcionamiento de los sistemas.

Cuando se dispone de un amplio registro de datos de funcionamiento en presencia de fallos, es factible utilizarlos para generar bases de casos o crear clasificadores de caja negra mediante técnicas de aprendizaje. También se pueden utilizar para generar clasificadores simbólicos, como árboles o conjuntos de reglas. Dado que se requieren numerosos ejemplos de cada posible fallo a considerar, estas técnicas se suelen aplicar en dominios en que el sistema objeto de diagnosis permanece estable el tiempo suficiente para acumular abundantes ejemplos de cada fallo. Si no fuese así, pero fuera posible simular de forma realista el funcionamiento del sistema en presencia de fallos, se puede recurrir a generar ejemplos de simulación para crear este tipo de modelos.

Cuando la mayor parte del conocimiento de diagnóstico está en manos de los especialistas que realizan el diagnóstico, es posible utilizar técnicas de Ingeniería de Conocimiento para construir modelos de clasificación simbólica. Esta ha sido la aproximación tradicional en los Sistemas Expertos para la diagnosis. Aunque parezca redundante, el requisito fundamental para la construcción de este tipo de modelos es la existencia de un cuerpo de experiencia y la disponibilidad de los expertos. Por consiguiente, esta técnica sólo es aplicable a aquellos dominios en que el sistema objeto de diagnóstico es lo suficientemente estable para permitir la creación de un cuerpo de experiencia. Un buen ejemplo de ello es el dominio de la medicina.

Si poseemos información sobre los principios de funcionamiento de los sistemas de diagnóstico, es posible crear modelos de comportamiento explícitos y utilizar la aproximación basada en modelos. Esto es particularmente cierto en aquellos dominios de la ingeniería en que disponemos de los modelos de diseño. También en aquellos dispositivos físicos en los que es posible elaborar modelos de comportamiento a partir de sus principios básicos de funcionamiento. Este tipo de modelos son los que más información aportan sobre el sistema y permiten elaborar sistemas de diagnosis con mayor capacidad de generación y discriminación de hipótesis. En principio, ni siquiera es preciso disponer de datos de funcionamiento ni experiencia previa para crear el sistema de diagnóstico: este se podría generar como parte del proceso de diseño del dispositivo. Lamentablemente, muchos dominios no permiten esta aproximación. De nuevo la diagnosis médica es un buen ejemplo de ello. Tampoco todos los sistemas artificiales permiten esta aproximación. La presencia de componentes con comportamientos complejos, difíciles de modelar, y la ocurrencia de múltiples interrelaciones entre las variables, que complica la estructura del sistema, son factores que dificultan la creación de modelos útiles para la diagnosis.

12.7 Resumen

De forma genérica, el proceso de diagnosis requiere razonar con el modelo del sistema para generar hipótesis de diagnosis que se prueban y discriminan utilizando observaciones. Como habitualmente no disponemos de todas las observaciones que nos permitirían discriminar entre las hipótesis de diagnóstico, el proceso de razonamiento se intercala con la actuación sobre el sistema para obtener nuevas observaciones o realizar pruebas.

Para intentar dar una visión unificada de las distintas aproximaciones a la tarea de diagnosis se ha utilizado un esquema genérico que identifica cuatro componentes básicos de un sistema de diagnosis: los espacios de **Datos**, **Hipótesis** y **Reparaciones**, junto al **Modelo del Sistema** a diagnosticar. Cada uno de estos espacios soporta operaciones de diagnosis que se apoyan en el modelo del sistema. Las operaciones básicas son **Monitorización** o detección de anomalías, **Generación**, **Prueba** y **Discriminación** de hipótesis. La forma de estos espacios y la índole de las operaciones básicas están fuertemente condicionadas por la naturaleza del modelo del sistema.

En este capítulo se han presentado tres tipos de modelos del sistema: **Árboles de Fallos**, **Modelos de Clasificación Simbólica** y **Modelos Basados en la Estructura y el Comportamiento**.

Los **árboles de fallos** son una aproximación práctica a la tarea de diagnosis. Más que modelar el sistema, codifican un protocolo de operación para guiar el proceso de diagnosis. Esto dificulta su mantenimiento y la generación de explicaciones del diagnóstico.

Los **modelos de clasificación simbólica**, esencialmente, capturan relaciones entre síntomas y fallos sobre las que se articulan las operaciones de diagnosis. Se suelen elaborar utilizando métodos de Ingeniería de Conocimiento, y su principal limitación es su fuerte dependencia del dispositivo a diagnosticar y la experiencia que de él se tiene.

Los **modelos basados en la estructura y el comportamiento** proporcionan la máxima información del modelo a diagnosticar y permiten el desarrollo de teorías formales de aplicación general. Su principal inconveniente radica en la dificultad de elaborar un modelo de utilidad para la diagnosis.

En la actualidad no se dispone de un método de diagnosis que sea aplicable de forma efectiva a cualquier tipo de problema. Ello explica la gran cantidad de métodos y modelos que se han propuesto para abordar esta tarea. Su estudio excede ampliamente los límites de un capítulo. La decisión final sobre qué método y modelo son los más apropiados para un problema concreto depende de la naturaleza del sistema y del conocimiento que se tiene de él.

En problemas complejos, es cada vez más frecuente la integración de técnicas de distinta naturaleza; distintos métodos se aplican a diferentes aspectos del problema, intentando obtener el máximo rendimiento de la información y el conocimiento disponibles de un sistema. Un ejemplo paradigmático es el sistema TIGER, que integra un sistema de diagnosis mediante clasificación simbólica junto a un sistema de diagnosis basado en consistencia; el primero se ocupa de fallos difíciles de modelar y que

requieren una actuación rápida, mientras el segundo realiza un análisis más detallado en un subsistema que puede ser abordado con estas técnicas.

Desde el punto de vista de las aplicaciones, la demanda de sistemas de diagnosis no deja de crecer. Las industrias de la automoción, aeronáutica y aeroespacial son particularmente activas en el desarrollo de sistemas de diagnosis; demandan sistemas de diagnosis empotrados en el propio vehículo (*diagnosis on-board*) y sistemas de diagnosis para la reparación y mantenimiento en el taller (*diagnosis off-board*). Encontramos aplicaciones en la industria de procesos, generación y distribución eléctrica, telecomunicaciones, redes de distribución de agua y en general en todas las aplicaciones donde los costes o riesgos asociados a la pérdida de funcionalidad de un sistema no son aceptables.

Desde el punto de vista de la investigación, la diagnosis es un área muy activa con numerosos frentes abiertos. El desarrollo de métodos basados en modelos, particularmente para sistemas dinámicos, tanto continuos como discretos, es uno de los campos que más atención recibe por parte de la comunidad investigadora. Se trabaja tanto en la propuesta de nuevos métodos como en el desarrollo de técnicas de modelado adecuadas para la diagnosis. El desarrollo y la aplicación de técnicas de aprendizaje para la obtención de modelos para la diagnosis también es un área de importante actividad. La diagnosis preventiva (para la detección de fallos incipientes o la anticipación de los mismos) y la diagnosis de sistemas distribuidos son dos campos de especial interés y que previsiblemente experimentarán un importante desarrollo en un futuro cercano.

12.8 Lecturas recomendadas

La obra de Jackson, [Jackson, 1999], constituye una buena introducción a los métodos de diagnosis mediante clasificación simbólica utilizados en los Sistemas Expertos; ilustra los distintos métodos con sistemas clásicos, proporciona una breve descripción de otras aproximaciones e incluye ejemplos prácticos de codificación.

Possiblemente, el libro de texto que trata de forma más detallada y sistemática la tarea de diagnosis es la obra de Stefik *Knowledge Systems*, [Stefik, 1995]. Dedica un capítulo completo a los métodos de clasificación simbólica y otro a la diagnosis. La presentación de los métodos de clasificación simbólica en este capítulo se inspira en esta obra. También es suya la propuesta de esquema genérico de sistema de diagnosis que se ha adoptado en este capítulo. Es una lectura obligada para todo aquel que quiera profundizar en la diagnosis desde la perspectiva de la IA.

Los modelos de clasificación por recubrimiento fueron introducidos por Reggia, [Peng y Reggia, 1990]. Es un texto de carácter formal que presenta estos modelos en profundidad y con mucho detalle.

Para aquel lector interesado en el origen y evolución de los métodos basados en conocimiento para la diagnosis, el libro *Rule-based Expert Systems. The MYCIN experiments of the Stanford heuristic programming project*, [Buchanan y Shortliffe, 1984], será de interés.

El libro de Price [Price, 1999] constituye una introducción muy asequible a la diagnosis. Se centra en los árboles de fallos, el paradigma de razonamiento basado en

casos y los métodos basados en modelos. Tiene un enfoque bastante práctico, trata temas que no se han incluido en este capítulo, como los FMEA (Análisis de efectos de modos de fallos), su generación automática a partir de modelos y herramientas software.

Los fundamentos de la aproximación DX a la diagnosis basada en modelos se asentaron a finales de los años ochenta. Los principales resultados se recogen en la obra *Readings in Model-based Diagnosis* [Hamscher y Console, 1992], una colección de artículos editados que contiene las referencias básicas de esta aproximación. Desde el punto de vista teórico, no se han producidos avances significativos en este campo. Sí es de destacar el gran esfuerzo realizado para aplicar estas técnicas a problemas reales y a sistemas complejos. La revista *IEEE Trans. on Systems, Man, and Cybernetics. Part B* [Biswas y otros, 2004], dedica un número especial a estas técnicas, que refleja el estado del arte hasta la fecha. Una introducción más asequible a la diagnosis basada en consistencia se encuentra en [Dressler y Struss, 1996]. Para conocer los últimos avances en esta disciplina, presentados de forma más sistemática que en una recopilación de artículos, es de interés el capítulo *Model-based Problem Solving*, [Struss, 2007], actualmente en edición.

Fuera del campo de la IA, posiblemente la comunidad de Ingeniería de Control es la que más atención ha dedicado a la diagnosis, especialmente de sistemas dinámicos. Esta aproximación a la diagnosis, conocida por las siglas FDI, es tratada en profundidad por [Patton y otros, 1989] y [Blanke y otros, 2003].

Hay dos conferencias dedicadas a la diagnosis de dispositivos físicos, donde se publican los últimos avances en este campo. La comunidad DX celebra anualmente el taller *International Workshop on Principles of Diagnosis*. La comunidad FDI celebra cada tres años la conferencia *Safe Process*. Los resultados más consolidados se publican en las principales conferencias de IA, como la ECAI, *European Conference on Artificial Intelligence*, IJCAI, *International Joint Conference on Artificial Intelligence*, o AAAI, *Conference of the American Association for the Advancement of Artificial Intelligence*.

12.9 Ejercicios Propuestos

12.1. Este ejercicio se basa en una versión simplificada y muy estilizada de un problema de diagnóstico en una sección de una planta azucarera. Presentaremos tres versiones, con distinto grado de elaboración. En esta primera versión el espacio de datos contiene sólo tres elementos: D_1 =“nivel del vaso del difusor oscila”, D_2 =“tendencia a bajar del nivel del vaso del difusor”, y D_3 =“tendencia a subir del nivel del vaso del difusor”. El espacio de soluciones contiene seis elementos: S_1 =“no se añade antiespumante”, S_2 =“parada de los rompe-espumas”, S_3 =“cabeza del difusor sucia”, S_4 =“tapón en el interior del difusor”, S_5 =“fallo severo bomba extracción”, y S_6 =“pérdida rendimiento bomba extracción”. S_1 y S_2 se presentan cuando se observa D_1 , con independencia de las observaciones de D_2 y D_3 . S_3 y S_4 se presentan cuando se observa D_2 con independencia de las observaciones de D_1 y D_3 . S_5 y S_6 se presentan cuando se observa D_3 y no se observa D_1 , con independencia de la observación de D_2 .

Elaborar un modelo de clasificación por recubrimiento que capture las relaciones que

se mencionan expresamente en esta descripción del problema. Responder, además, a las siguientes cuestiones:

1. ¿Qué candidatos son rechazados, cuáles son consistentes y cuáles son explicaciones para el patrón de observaciones $<1, 0, 0>$?
2. Lo mismo si el patrón de observaciones es $<0, ?, 1>$.
3. Suponiendo hipótesis de fallo único y consistencia como criterio de inclusión, obtener las soluciones del problema de diagnóstico si el patrón de observaciones es $<1, 1, 0>$.

12.2. Presentamos ahora una segunda versión del mismo problema de diagnóstico. El espacio de soluciones es el mismo. Con el objetivo de aumentar la capacidad de discriminar entre distintas soluciones, el espacio de datos se amplía con: D_4 =“parada bomba antiespumante”, D_5 =“parada rompe-espumas”, D_6 =“intensidad motor cabeza difusor alta”, D_7 =“intensidad motor difusor alta”, D_8 =“caudal extracción nulo”. Junto a las condiciones exigidas en el problema anterior, ahora se añaden: S_1 requiere la presencia de D_4 , y S_2 la de D_5 . S_3 requiere la presencia de D_6 , y S_4 la de D_7 . S_5 requiere la presencia de D_8 , y S_6 su ausencia.

Elaborar un nuevo modelo de recubrimiento secuencial que capture la información que se ha añadido.

12.3. El modelo de recubrimiento elaborado en el problema anterior se puede transformar en un modelo de clasificación jerárquico que, al aportar estructura, facilita la comprensión y el mantenimiento del modelo. Para ello, basta con añadir tres soluciones abstractas: SA_1 =“problemas de espuma”, SA_2 =“problemas circulación difusor”, SA_3 =“problemas extracción vaso difusor”. SA_1 tiene como hijos a S_1 y S_2 ; SA_2 a S_3 y S_4 ; SA_3 a S_5 y S_6 . Las relaciones de las soluciones abstractas con los datos se pueden obtener a partir de la primera versión del problema. Las relaciones que permiten refinar las soluciones abstractas se pueden obtener de la información que aporta la segunda versión del problema.

Elaborar un modelo de clasificación jerárquico conforme al planteamiento que se propone en esta tercera versión del problema.

12.4. Utilizar el método de diagnosis mediante clasificación jerárquica guiada por datos con el modelo elaborado en el problema anterior y con el siguiente patrón de observaciones $OBS = <0, 1, 0, ?, ?, 1, 0, 0>$. Suponer que el procedimiento *Monitor* devuelve $OBS,2 = 1$. Asumir que las observaciones están al nivel de abstracción adecuado, de modo que el procedimiento *Abstraer* no las modifica. Tener en cuenta que el método de diagnosis no dispone del patrón de observaciones OBS , sino que solicita alguno de sus componentes a medida que los necesita.

Obtener el conjunto de soluciones que encuentra el algoritmo.

12.5. Considerar el ejemplo del sistema multicaja utilizado para ilustrar la diagnosis basada en consistencia. En este problema se propone un modelo abstracto de este sistema, adaptado de [Bakker y otros, 1989], en el cual el dominio de todas las variables

es *Bien*, *Mal*, ?. *Bien* indica que la observación es la esperada, *Mal* que se aparta del valor esperado y ? que desconocemos su valor. El comportamiento correcto de cualquier componente, COMP, se modela con la siguiente regla:

Si para toda variable de entrada, var_e de COMP, $var_e = \text{Bien}$,
Entonces para toda variable de salida, var_s de COMP, $var_s = \text{Bien}$.

1. Obtener los conflictos y diagnosis minimales para el siguiente conjunto de observaciones $A = [\text{Bien}]$, $B = [\text{Bien}]$, $C = [\text{Bien}]$, $D = [\text{Bien}]$, $E = [\text{Bien}]$, $F = [\text{Mal}]$, $G = [\text{Bien}]$.
2. Comparando los resultados obtenidos por el modelo detallado con los obtenidos por este modelo abstracto, ¿qué se puede decir de los conflictos minimales? ¿Son correctos los candidatos propuestos?

12.6. El modelo abstracto del sistema multicaja tiene muy poca capacidad de diagnóstico, debido principalmente a que no permite propagar valores *Mal* a través de los componentes. Esta limitación se puede atenuar introduciendo la siguiente regla:

Si para alguna variable de entrada, var_e de COMP, $var_e = \text{Mal}$,
Entonces para la variable de salida, var_s de COMP, $var_s = \text{Mal}$.

Hay que ser cuidadoso en la interpretación de esta regla y considerar como variable de entrada todas aquellas variables cuyo valor se conoce, aplicándola cuando se conocen todos los valores de todas las variables menos una.

1. Obtener los conflictos y diagnosis minimales para el siguiente conjunto de observaciones $A = [\text{Bien}]$, $B = [\text{Bien}]$, $C = [\text{Bien}]$, $D = [\text{Bien}]$, $E = [\text{Bien}]$, $F = [\text{Mal}]$, $G = [\text{Bien}]$.
2. Comparando los resultados obtenidos por el modelo detallado con los obtenidos por este segundo modelo abstracto, ¿qué se puede decir de los conflictos minimales? ¿Son correctos los candidatos propuestos?

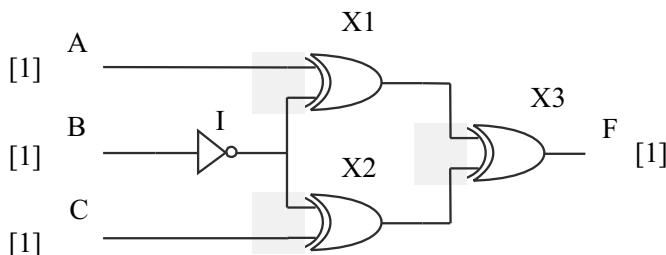


Figura 12.30: Sistema digital.

12.7. Este problema ilustra el hecho de que la propagación local de valores obtenidos a partir de modelos locales de componentes no siempre es suficiente para obtener las

propiedades globales de un sistema. Considerar el sistema digital descrito en la Figura 12.30. Está formado por tres puertas or exclusivo, X_1 , X_2 y X_3 , y por un inversor, I . Responder a las siguientes cuestiones.

1. Obtener los conflictos y diagnosis minimales con las observaciones que proporciona la figura.
2. Suponer que ahora observamos $B = [0]$, manteniéndose constantes las restantes observaciones. Obtener los nuevos conflictos y diagnosis minimales.
3. A la vista de los resultados obtenidos en los dos apartados anteriores, y para las observaciones de A , C y F proporcionadas por la Figura 12.30, ¿debería ser $[I]$ un candidato? ¿Qué conflicto minimal deberíamos obtener?

Referencias

- BAKKER, R.; VAN SOEST, D.; HOGENHUIS, P. y MARS, N.: «Fault Models in Structural Diagnosis». En: *International Workshop of Model-based Diagnosis*, Paris, 1989.
- BALAKRISHNAN, K. y HONAVAR, V.: «Intelligent diagnosis systems». *Journal of Intelligent System*, 1998, **8(3-4)**, pp. 239–290.
- BISWAS, G.; CORDIER, M. O.; LUNZE, J.; TRAVÉ-MASSUÈS, L. y M., STAROSWIECKI: «Diagnosis of complex systems: bridging the methodologies of the FDI and DX communities». *IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics. Special Issue on Diagnosis of Complex Systems*, 2004, **34(5)**, pp. 2159–2162.
- BLANKE, M.; KINNAERT, M.; LUNZE, J. y STAROSWIECKI, M.: *Diagnosis and Fault Tolerant Control*. Springer-Verlag, 2003.
- BROWN, J.S.; BURTON, R.R. y DE KLEER, J.: «Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III». En: *Intelligent Tutoring Systems*, pp. 230–279. Sleeman, D. and Brown, j. S. (Eds). Academic Press, Boston, 1982.
- BUCHANAN, B. G. y SHORTLIFFE, E. H.: *Rule-based Expert Systems. The MYCIN experiments of the Stanford heuristic programming project*. Addison Wesley, Reading, 1984.
- CLANCEY, W. J.: «Heuristic classification». *Artificial Intelligence*, 1985, **27(3)**, pp. 289–350.
- CONSOLE, L.: «Model-based Diagnosis history and state of the art». En: *MONET Summer School. MONET Network*, , 2000.
<http://monet.aber.ac.uk:8080/monet/index.html>
- DAVIS, R.: «Expert Systems: Where are we? and where do we go from here?» *Artificial Intelligence Magazine*, 1982, **3(2)**, pp. 3–22.
- DE KLEER, J. y WILLIAMS, B. C.: «Diagnosis with Behavioral Modes.» En: *Proceedings of the 11th International Joint Conference on AI (IJCAI-89)*, , 1989.
- DE KLEER, J. y WILLIAMS, B.C.: «Diagnosing multiple faults». *Artificial Intelligence*, 1987, **32(1)**, pp. 97–130.
- DRESSLER, O. y STRUSS, P.: «The Consistency-based Approach to Automated Diagnosis of Devices». En: ed. Gerhard Brewka (Ed.), *Principles of Knowledge Representation*, CSLI Publications. Standford, 1996.
- GENESERETH, M. R: «The Use of Design Descriptions in Automated Diagnosis». *Artificial Intelligence*, 1984, **24(1-3)**, pp. 411–436.
- HAMSCHER, W. y CONSOLE, J., L. AND. DE KLEER: *Readings in Model-based Diagnosis*. Morgan-Kaufmann Pub., San Mateo., 1992.

- HAMSCHER, W. C.: «Modeling digital circuits for troubleshooting». *Artificial Intelligence*, 1991, **51(1-3)**, pp. 223–272.
- HAMSCHER, WALTER: «Temporally Coarse Representation of Behavior for Model-based Troubleshooting of Digital Circuits.» En: *IJCAI*, pp. 887–893, 1989.
- JACKSON, P.: *Introduction to Expert Systems*. Addison Wesley, Reading., 1999.
- PATTON, R.; FRANK, P. y CLARK, R.: *Fault diagnosis in dynamic systems. Theory and applications*. Prentice Hall International, 1989.
- PENG, Y. y REGGIA, J.: *Abductive Inference Models for Diagnostic Problem Solving*. Springer Verlag New York, 1990.
- PRICE, C.: *Computer-based diagnostic systems*. Springer Verlag, Nueva York, 1999.
- REITER, R.: «A theory of diagnosis from first principles». *Artificial Intelligence*, 1987, **32**, pp. 57–95.
- STEFIK, M.: *Introduction to Knowledge Systems*. Morgan-Kaufmann Pub., San Mateo, 1995.
- STRUSS, P.: *Model-based Problem Solving. Handbook of Knowledge Representation*. Elsevier, 2007.
- STRUSS, P. y DRESSLER, O.: «Physical negation: Integrating fault modes into the general diagnostic engine». En: *Proc. 11th Int. Joint Conf. on Artificial Intelligence*, pp. 1318–1323. Morgan Kaufmann Publishers, 1989.
- TRAVÉ-MASSUYÈS, L. y MILNE, R.: «Gas-Turbine Condition Monitoring Using Qualitative Model-based Diagnosis». *IEEE Expert*, 1997, **12(3)**, pp. 22–31.
- VENKATASUBRAMANIAN, V.; RAGHUNATHAN, R.; KEWEN, Y. y KAVURI, S.N.: «A review of Process fault detection and diagnosis. Part I. Part II. Part III». *Computers & Chemical Engineering*, 2002, **27**, pp. 293–346.
- WEISS, S.M.; KULIKOWSKI, C.A.; AMAREL, S. y SAFIR, A.: «A model-based method for computer-aided medical decision making». *Artificial Intelligence*, 1978, **11**, pp. 145–172.

Capítulo 13

Planificación

Eva Onaindia de la Rivaherrera y Antonio Garrido Tejero
Universidad Politécnica de Valencia

13.1 Introducción

El campo de la **planificación en IA** tiene como objetivo construir algoritmos de control que permitan a un agente sintetizar una secuencia de acciones que le lleve a alcanzar sus objetivos. Un problema de planificación en IA es un problema de búsqueda que requiere encontrar una secuencia eficiente de acciones para conducir a un sistema desde un estado inicial hasta un estado objetivo.

Consideremos un escenario formado por un conjunto de ciudades donde el objetivo de un agente es transportar un paquete de una ciudad origen c_0 a una destino c_D . El algoritmo de búsqueda tendría que examinar todas las posibles rutas (planes) que se pueden formar para ir de c_0 a c_D . Para un caso concreto de $n = 7$ ciudades con cinco ciudades intermedias, el espacio de búsqueda estaría formado por 380 rutas diferentes, y para tamaños mayores de problemas este espacio podría resultar inabordable ($(\sum_{j=1}^n \prod_{i=j}^n i) + 1$ rutas).

A priori, y dado el planteamiento inicial de este problema, podrían aplicarse técnicas clásicas de resolución de problemas para encontrar una solución, como por ejemplo la aplicación del algoritmo del viajante de comercio. Sin embargo, la complejidad en problemas reales de planificación puede llegar a ser tan elevada que se hace necesario un razonamiento inteligente sobre las acciones y sus consecuencias para afrontar eficientemente la resolución de estos problemas. Supongamos que el agente dispone de tres camiones de diferente tamaño para transportar el paquete; en este caso, el espacio de búsqueda del agente se incrementaría hasta $3^*380 = 1140$ posibles rutas. Adicionalmente, si en el estado inicial hay más de un paquete a transportar de c_0 a c_D habría que considerar todas las posibles formas de transportar los paquetes, bien individualmente o bien en grupos, utilizando uno, dos o todos los camiones a través de todas las posibles rutas. Al crecer el espacio de búsqueda de forma exponencial, los algoritmos tradicionales de búsqueda no resultan viables para resolver estos problemas. Si se introduce un factor adicional en nuestro escenario, como la existencia de

más paquetes en una ciudad intermedia cuyo destino es igualmente cD, un planificador inteligente deberá razonar sobre qué camiones utilizar para transportar los paquetes y qué ruta(s) realizar para cumplir el objetivo. Existen múltiples alternativas, como: 1) utilizar un único camión para todos los paquetes del problema y planificar una única ruta que pase por la ciudad intermedia, 2) utilizar dos camiones, uno para los paquetes que están en c0 y otro para los paquetes de la ciudad intermedia (en este caso habría que planificar rutas independientes, una para cada camión), 3) utilizar todos los camiones disponibles, etc.

Como en todo problema de búsqueda inteligente, un elemento clave en planificación es disponer de buenas **funciones heurísticas** para guiar eficientemente la búsqueda del planificador (véase el capítulo 9). En nuestro problema, se daría mayor prioridad a un estado en el que existe un camión en la ciudad intermedia porque en dicho estado se satisfacen parte de las condiciones necesarias para transportar los paquetes de esa ciudad. Esto indica que dicho estado estará más próximo a un estado final donde se satisfagan todos los objetivos del problema y, por tanto, el proceso de búsqueda a partir de ese estado necesitará menos acciones o pasos de ejecución para alcanzar el objetivo. Uno de los propósitos de la planificación es el desarrollo de funciones heurísticas *independientes del dominio*; es decir, funciones genéricas y reutilizables que pueden aplicarse al espacio de búsqueda de cualquier problema, con independencia del tipo de escenario en el que se desarrolla dicho problema.

Los problemas de planificación se suelen plantear en sistemas dinámicos donde, dado el estado actual y los objetivos, deducir la siguiente acción a aplicar no es una tarea obvia. La planificación es una tarea compleja y ésta es la razón por la cual la mayoría de planificadores trabajan sobre un modelo restringido del entorno (**planificación clásica**), siendo dicho modelo determinista, estático y totalmente observable. Concretamente, la planificación clásica fija las siguientes asunciones:

- A1) **Modelo observable.** El mundo es totalmente observable y no existe información desconocida para el planificador.
- A2) **Modelo estático.** Se puede predecir correctamente la evolución de las secuencias de acciones que se aplican sobre un estado inicial completamente conocido, ya que no hay influencias externas que afecten al entorno. Los objetivos son conocidos antes de comenzar la planificación y no cambian durante el transcurso del proceso de planificación.
- A3) **Modelo determinista.** Los efectos de la aplicación de una acción en un estado son totalmente predecibles, lo que conduce determinísticamente a un único estado.
- A4) **Enfoque proposicional.** Todas las variables del modelo de planificación (sentencias atómicas) pertenecen al dominio lógico, tomando por tanto los valores *cierto* o *falso*.
- A5) **Acciones instantáneas.** Todas las acciones del modelo de planificación tienen la misma duración y se consideran atómicas e instantáneas.

- A6) **Acciones no descomponibles.** Las acciones del modelo son directamente ejecutables en el entorno y no pueden descomponerse o dividirse en subacciones.
- A7) **Planificación offline.** La tarea de planificación consiste en construir un plan completo que satisface el objetivo antes de la ejecución de cualquier parte del mismo.

A pesar de estas simplificaciones, la resolución de un problema de planificación es PSPACE-completo. Por ello no resulta factible el empleo de técnicas clásicas de resolución de problemas o demostración de teoremas. Este capítulo está dedicado a presentar los principios fundamentales de la planificación como una actividad centrada en técnicas de búsqueda, representación y tratamiento de un problema de planificación, algoritmos y técnicas de control para hacer la búsqueda más eficiente.

13.2 Problema de planificación

Un problema de planificación se describe mediante un conjunto de acciones, un estado inicial del entorno y una descripción del objetivo a conseguir. Este problema se resuelve obteniendo el conjunto de acciones que transforman el estado inicial en un estado que satisface el objetivo. Esta visión del problema de planificación, totalmente centrada en el concepto de acción, hereda muchos aspectos del cálculo de situaciones desarrollado por McCarthy en [McCarthy y P.J., 1969], en donde se especifica cómo las situaciones, descritas en un lenguaje de primer orden, se ven afectadas por las acciones ejecutadas por un agente. Para lograr un proceso de planificación eficiente es tan importante contar con buenos algoritmos como con buenos lenguajes de modelado y representación. El sistema STRIPS ha condicionado la gran mayoría de trabajos sobre planificación desde comienzos de los años 70, gracias a la definición de una sencilla sintaxis para la especificación de problemas de planificación.

Un problema de planificación STRIPS se define como una tripleta $\mathcal{P} = \langle \mathcal{I}, \mathcal{G}, \mathcal{O} \rangle$ donde \mathcal{I} es el estado inicial del problema, \mathcal{G} es el objetivo (*goal*) a conseguir y \mathcal{O} es el conjunto de operadores de planificación. Un **estado** de un problema de planificación es una representación del conjunto de propiedades o características de los objetos que intervienen en el problema junto con los valores de dichas propiedades. En STRIPS se trabaja con literales de primer orden y las descripciones de estados se componen de literales positivos totalmente instanciados y sin dependencias funcionales. En un dominio de transporte, como el indicado en la sección anterior, donde existen paquetes, camiones y ciudades, un estado del problema representaría la ubicación de paquetes y camiones en ciudades. Un predicado binario de primer orden como *pos(?obj, ?ciu)* se utiliza para indicar la posición de cualquier objeto del problema (paquete o camión) en una ciudad determinada. En la Tabla 13.1 se representa el estado inicial \mathcal{I} de un problema de transporte en el que existe un paquete *p1* y un camión *c1*, ambos localizados en la ciudad *ca*. Se asume la hipótesis de *mundo cerrado*, lo que significa que sólo las condiciones que se listan explícitamente tienen valor *cierto* y que cualquier condición que no se menciona en un estado se considera *falsa*.

Estado inicial (\mathcal{I})	$\text{pos}(\text{p1}, \text{cA}) \wedge \text{pos}(\text{c1}, \text{cA})$
Objetivo (\mathcal{G})	$\text{pos}(\text{p1}, \text{cB})$
Operadores (\mathcal{O})	$\text{mv}(\text{?cam}, \text{?ori}, \text{?des})$ Pre: $\text{pos}(\text{?cam}, \text{?ori})$ Efe: $\text{pos}(\text{?cam}, \text{?des}) \wedge \neg \text{pos}(\text{?cam}, \text{?ori})$ $\text{cg}(\text{?paq}, \text{?cam}, \text{?ciu})$ Pre: $\text{pos}(\text{?paq}, \text{?ciu}) \wedge \text{pos}(\text{?cam}, \text{?ciu})$ Efe: $\text{en}(\text{?paq}, \text{?cam}) \wedge \neg \text{pos}(\text{?paq}, \text{?ciu})$ $\text{dcg}(\text{?paq}, \text{?cam}, \text{?ciu})$ Pre: $\text{pos}(\text{?cam}, \text{?ciu}) \wedge \text{en}(\text{?paq}, \text{?cam})$ Efe: $\text{pos}(\text{?paq}, \text{?ciu}) \wedge \neg \text{en}(\text{?paq}, \text{?cam})$

Tabla 13.1: Definición de un problema simple de transporte que se empleará a lo largo del capítulo utilizando STRIPS.

Para describir el **objetivo** de un problema de planificación hay que especificar el valor que se desea que tengan las propiedades de los objetos en el estado meta del problema. A diferencia del estado inicial, un objetivo de planificación no es una descripción completa de un estado sino que describe simplemente el valor de aquellas propiedades de objetos que constituyen el objetivo del problema. En la Tabla 13.1 se representa el objetivo \mathcal{G} para el problema de transporte, que consiste en tener el paquete p1 en la ciudad cB . Se puede observar que no se especifica la posición final del camión c1 , indicando así que éste puede finalizar en cualquier ciudad.

Un **operador** es una generalización de una función de transición que toma como entrada un estado del problema y determina cuál será el siguiente estado; una **acción** es una instanciación de un operador. En la Tabla 13.1 se muestran los tres operadores \mathcal{O} para el problema de transporte. Un operador consta de tres partes:

- Nombre y lista de argumentos. Por ejemplo, el primer operador es mv (mover un camión de una ciudad a otra) y requiere tres argumentos: el camión ?cam a mover, la ciudad origen ?ori del camión y la de destino ?des .
- Precondiciones (Pre). Conjunto de literales positivos que determinan las condiciones que deben satisfacerse en el estado anterior a la ejecución del operador y mantenerse durante toda la ejecución del mismo. Las variables que aparecen en las precondiciones deben aparecer también en la lista de argumentos. Por ejemplo, las precondiciones del operador cg (cargar un paquete en un camión) son que el paquete y el camión se encuentren ambos en la ciudad ?ciu .
- Efectos (Efe). Conjunción de literales que describe cómo cambia el estado cuando se ejecuta el operador. Un literal positivo es un efecto que se añade en el estado resultante (literal con valor *cierto* en el nuevo estado), y un literal negativo (\neg) es un efecto que se elimina (literal con valor *falso* en el nuevo estado). La lista de variables de los efectos también debe aparecer en la lista de argumentos. Por ejemplo, los efectos del operador dcg (descargar un paquete de un camión en una ciudad) son que el paquete se encontrará en la ciudad ?ciu y dejará de estar dentro del camión ($\neg \text{en}(\text{?paq}, \text{?cam})$).

Una acción, como por ejemplo $\text{mv}(c1, cA, cB)$, se dice que es **aplicable** en un estado si se satisfacen las precondiciones de la acción en dicho estado. Las precondiciones de la acción anterior ($\text{pos}(c1, cA)$) se satisfacen en el estado inicial de la Tabla 13.1, por lo que la acción es aplicable en dicho estado, dando como resultado un nuevo estado en el que se añade el literal $\text{pos}(c1, cB)$ y se elimina el literal $\text{pos}(c1, cA)$. Sin embargo, esta misma acción no es aplicable en el estado $\mathcal{E} = \{\text{pos}(c1, cB), \text{pos}(p1, cA)\}$. Si una acción no es aplicable no produce ningún efecto. En conclusión, la aplicación de una acción en un estado \mathcal{E} produce un nuevo estado $\mathcal{E}' = \mathcal{E} - \text{Efe}^- \cup \text{Efe}^+$, donde Efe^+ son los efectos positivos que se añaden y Efe^- son los efectos negativos que se borran, respectivamente. Una de las grandes aportaciones de STRIPS es la asunción para evitar la complejidad del *problema marco* introducida en [McCarthy y P.J., 1969], la cual consiste en asumir que los únicos cambios que se producen como resultado de la aplicación de una acción son aquellos literales que explícitamente se mencionan como efectos de la misma; es decir, el resto de literales se satisface también en el nuevo estado.

Por último, la solución a un problema de planificación se denomina **plan**. En su forma más simple, un plan es una secuencia de acciones que, cuando se ejecuta desde el estado inicial, produce un estado que satisface el objetivo. El plan solución al problema de la Tabla 13.1 sería $\langle \text{cg}(p1, c1, cA), \text{mv}(c1, cA, cB), \text{dcg}(p1, c1, cB) \rangle$.

13.3 Lenguaje de planificación PDDL

El lenguaje STRIPS se planteó con el deseo de diseñar algoritmos simples y eficientes, sin complicar excesivamente la definición de problemas reales. Con el objetivo de enriquecer la expresividad permitida por STRIPS, en los últimos años se han desarrollado otros lenguajes de planificación entre los que destaca PDDL, que últimamente se ha convertido en un estándar en planificación. Actualmente existen varias versiones de PDDL, siendo la más reciente PDDL3 que engloba todas las características de sus predecesoras e incluye nuevas funcionalidades. En esta sección se apuntan sólo aquellas características de PDDL que introducen un cambio en el modelo semántico de las acciones de STRIPS.

- Acciones con duración (acciones *durativas*), donde precondiciones y efectos se asocian a los puntos de inicio y final de la duración de la acción; además, las precondiciones también se pueden asociar a todo el intervalo de duración. De esta forma se puede modelar mejor la *física* del problema. Por ejemplo, en el caso del operador mv , el camión deja de estar en $?ori$ en cuanto inicia su recorrido, y no llega a $?des$ hasta el final del mismo. Por tanto, el efecto negativo $\neg\text{pos}(\text{?cam}, ?ori)$ se produciría al iniciar el operador, y el efecto positivo $\text{pos}(\text{?cam}, ?des)$ al final del mismo. En la sección 13.8.1.1 se estudiará este modelo con más detalle dentro de la planificación temporal.
- Expresiones y variables numéricas. Las expresiones numéricas se construyen mediante operadores aritméticos y funciones numéricas, las cuales asocian valores numéricos a tuplas de objetos del problema. Las condiciones numéricas

en las acciones son siempre comparaciones entre pares de expresiones numéricas, mientras que los efectos permiten modificar los valores de las funciones numéricicas. Gracias al uso de variables numéricas se puede, por ejemplo, controlar el consumo de combustible (en litros) de un camión, comprobando que el camión dispone de cantidad suficiente para realizar un recorrido determinado (precondición numérica) y actualizando dicho valor al final de la ejecución de la acción (efecto numérico). En la sección 13.8.1.2 se estudiará este modelo con más detalle dentro de la planificación numérica.

- Métricas del problema. Permiten definir funciones de optimización como una combinación lineal de uno o varios factores del problema. Por ejemplo, se puede definir una métrica para minimizar el consumo total de combustible de todos los camiones de un problema, o una función para minimizar el tiempo total del plan y el consumo de energía de un robot, ponderando cada factor con un porcentaje determinado (véase la sección 13.8.1.2).
- Ventanas temporales. Se utilizan como una forma restrictiva de expresar eventos exógenos que no se ven afectados por las acciones del plan; es decir, hechos que serán *ciertos* o *falsos* en puntos de tiempo conocidos por el planificador. Por ejemplo, mediante el uso de ventanas temporales se puede expresar que el almacén donde se guardan los paquetes está abierto desde las 8h hasta las 20h. Para ello, se informa al planificador de la existencia de un hecho *cierto* (apertura del almacén) en el instante 8, y que dicho literal será *falso* a partir del instante 20. Estos hechos suceden en el plan independientemente de las acciones del mismo.
- Restricciones duras y blandas sobre el plan. Definen restricciones sobre la estructura de los planes que deben satisfacerse en los estados intermedios del plan. Estas restricciones se expresan mediante operadores modales como **always**, **sometime**, **at-most-once**, **at-end**, **always-within**, **hold-after**, etc. para indicar cómo y cuándo deben satisfacerse. Una restricción dura es aquélla que *obligatoriamente* debe cumplirse en el plan, como por ejemplo que todos los camiones deben visitar cada ciudad a lo sumo una vez. Una restricción blanda o preferencia es una restricción que debe satisfacerse en la medida de lo posible pero no es condición imprescindible para la obtención del plan. De este modo, algunas preferencias podrían no satisfacerse bien porque ello supone un coste excesivo que afecta a la función de optimización del problema o bien porque dicha preferencia entra en conflicto con otras restricciones u objetivos. Un ejemplo de preferencia sería: todos los camiones del problema deben finalizar, *preferiblemente*, en la ciudad **CA**.

Sopportar estas características permite que PDDL sea un lenguaje de modelado útil para la definición de problemas más cercanos a la realidad, haciendo hincapié no sólo en los elementos clave para conseguir un plan, sino también en los necesarios para especificar los criterios de calidad del plan.

13.4 Planificación en un espacio de estados

El enfoque más sencillo en planificación consiste en realizar una búsqueda en un espacio de estados (véase el capítulo 8), donde se genera un árbol de búsqueda cuyos nodos denotan estados, entendiendo estado como un conjunto de literales. Dado que las descripciones de acciones especifican tanto las precondiciones como los efectos, es posible realizar una búsqueda en ambas direcciones: búsqueda hacia delante desde el estado inicial o búsqueda hacia atrás desde los objetivos.

13.4.1 Búsqueda hacia delante

En la búsqueda hacia delante o **búsqueda progresiva** los nodos del árbol representan **estados** del problema de planificación, siendo el nodo raíz del árbol el estado inicial del problema. Un nodo se expande a partir de todas las acciones del problema que son **aplicables** en dicho estado, generando así el nuevo conjunto de nodos (estados) sucesores, los cuales formarán parte de la lista *ABIERTA* del árbol. Sea \mathcal{I} el estado inicial y $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ el conjunto de todas las posibles acciones del problema. El subconjunto de acciones aplicables en \mathcal{I} será $\{\mathbf{a}_i\} \mid \text{Pre}(\mathbf{a}_i) \in \mathcal{I}$. Los estados resultantes de aplicar las acciones correspondientes serán aquellos nodos que se formen eliminando del estado padre los efectos negativos y añadiendo los efectos positivos de la acción correspondiente; para el caso del estado \mathcal{I} , los sucesores que se forman serán: $\{\mathcal{E}_i\} \mid \mathcal{E}_i = \mathcal{I} - \text{Efe}^-(\mathbf{a}_i) \cup \text{Efe}^+(\mathbf{a}_i)$.

En un planificador progresivo la aplicación de cada acción genera un paso, y un plan consiste en una simple lista de pasos; se comienza desde la descripción del estado inicial y se busca el objetivo en el espacio de estados del problema. El plan se construye añadiendo pasos (acciones) totalmente instanciados al final del plan. Así, cuando se alcanzan los objetivos se devuelve una solución (plan) totalmente ordenada. Por este motivo, a los planificadores basados en este tipo de búsqueda también se les denomina planificadores de **orden total**, es decir sólo se exploran secuencias lineales de acciones que conectan el estado inicial del problema con el objetivo.

Durante la expansión del árbol, los nodos sucesores que se generan se introducen en la lista *ABIERTA* del árbol, los cuales constituyen el conjunto de estados candidatos a ser expandidos en la siguiente iteración. Dependiendo de la estrategia de búsqueda utilizada (anchura, profundidad, profundización iterativa, coste uniforme, búsqueda heurística, etc.) los nodos se ordenarán en la lista de distinto modo, siendo siempre el primer nodo de la lista el siguiente nodo a expandir (véase el Algoritmo 8.1 del capítulo 8). En el caso que nos atañe no emplearemos ninguna estrategia de búsqueda concreta, sino que el nodo a expandir se seleccionará no determinísticamente y se extraerá de la lista *ABIERTA* del árbol.

Ejemplo 13.1. En la Figura 13.1 se muestra el árbol de búsqueda que se genera para el problema de transporte de ejemplo. Las dos acciones aplicables en el estado inicial \mathcal{I} son aquéllas cuyas precondiciones se satisfacen en dicho estado ($\text{mv}(c1, cA, cB)$ y $\text{cg}(p1, c1, cA)$). Los estados resultantes de la aplicación de estas dos acciones se generan añadiendo los efectos positivos y borrando los efectos negativos de las mismas,

y dichos nodos se introducen en la lista ABIERTA. El número asociado a cada nodo indica un orden de expansión arbitrario de los nodos de la lista ABIERTA; de este modo, cuando se expande el nodo 2, la única acción aplicable es la **acción inversa** a la que ha generado el nodo 2 y, por consiguiente, el sucesor que se generaría sería un estado repetido que coincidiría con el estado \mathcal{I} . Lo mismo sucede en la expansión del nodo 3 con la aplicación de la acción $dcg(p1, c1, cA)$, que daría de nuevo un estado idéntico al estado \mathcal{I} , o con la aplicación de la acción $mv(c1, cB, cA)$ en la expansión del nodo 4. El objetivo $\mathcal{G} = \{\text{pos}(p1, cB)\}$ se satisface en el momento en que se selecciona el nodo 5 para ser expandido.

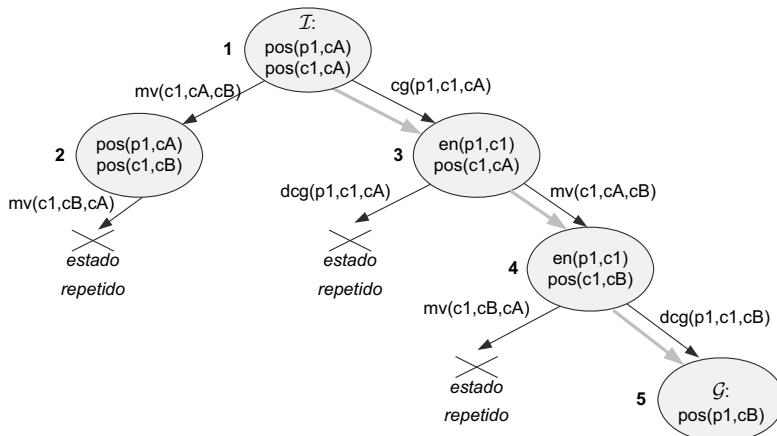


Figura 13.1: Ejemplo del árbol de búsqueda hacia delante que se generará para el problema de transporte (véase la Tabla 13.1). En trazo gris se muestra la rama que constituye el plan.

La búsqueda hacia delante fue la primera aproximación que se utilizó para resolver problemas de planificación. Este tipo de búsqueda plantea los siguientes inconvenientes:

- Se consideran todas las acciones aplicables en un estado sin desestimar aquellas acciones que son irrelevantes para la consecución de los objetivos. Por ejemplo, supongamos un problema con tres paquetes $\{p1, p2, p3\}$ y cuyo objetivo es simplemente llevar $p1$ a la ciudad cB . Una estrategia de búsqueda hacia delante considerará todas las acciones aplicables, incluyendo las de cargar, mover y descargar los paquetes $p2$ y $p3$, aunque éstas no influyan en la consecución del objetivo.
- Tal y como se indica en la Figura 13.1, en dominios *reversibles* donde existen acciones inversas que conducen de nuevo al estado antecesor, se producen

infinidad de ciclos en el árbol de búsqueda. Este es un factor que debe controlarse convenientemente para evitar un proceso de búsqueda infructuoso. Por este motivo, y porque el mismo estado puede alcanzarse por diferentes caminos (diferentes secuencias de acciones), en general se suele decir que se tiene un grafo de búsqueda en lugar de un árbol de búsqueda.

- El factor de ramificación para problemas de gran tamaño puede resultar en una explosión del espacio de búsqueda. Considérese un problema en el que hay 10 ciudades, 50 camiones y 20 paquetes a transportar. El objetivo es mover los 20 paquetes de la ciudad cA a la ciudad cB . Una solución sencilla es cargar los 20 paquetes en un camión, mover el camión de cA a cB y descargar los paquetes. Pero encontrar una solución puede resultar complejo porque el factor de ramificación es excesivamente elevado: cada uno de los 50 camiones puede ir a las 9 ciudades restantes, y cada uno de los 20 paquetes puede cargarse en cualquier camión en la ciudad donde éste se encuentre (si no está cargado) o descargarse (si está cargado). En general, el factor de ramificación de un planificador progresivo viene determinado por el número de acciones aplicables en cada estado. Si llamamos a este factor b y, teniendo en cuenta que la profundidad de la búsqueda, n , viene determinada por el mínimo número de acciones necesarias para alcanzar el objetivo desde el estado inicial, tenemos que el coste del árbol de búsqueda o número de nodos generados sería $O(b^n)$.

Por las razones expuestas arriba, se consideró que la búsqueda hacia delante en un espacio de estados era una estrategia muy poco eficiente para la resolución de problemas de planificación. Sin embargo, esta consideración ha cambiado recientemente porque la investigación en planificación de los últimos años ha dado luz a planificadores de búsqueda hacia delante muy eficientes gracias al empleo de excelentes funciones heurísticas. Evidentemente, sin el uso de buenas funciones heurísticas esta estrategia resulta inadecuada para resolver problemas de gran tamaño.

13.4.2 Búsqueda hacia atrás

Este tipo de búsqueda, también conocida como **búsqueda regresiva**, parte del objetivo \mathcal{G} del problema, el cual está constituido por el conjunto de literales a conseguir $\mathcal{G} = \{g_1, g_2, \dots, g_n\}$ y aplica inversamente los operadores del problema, generando así estados subobjetivos. El proceso terminará si se produce un conjunto de subobjetivos que se satisface en \mathcal{I} . El esquema de funcionamiento de la búsqueda hacia atrás se muestra en el Algoritmo 13.1. Inicialmente se parte de un plan vacío Π que se va construyendo sucesivamente en cada iteración del algoritmo. El objetivo de la búsqueda hacia atrás consiste en ir actualizando sucesivamente el conjunto \mathcal{G} (que inicialmente se compone de todos los objetivos del problema) hasta que se llega a un conjunto que se satisface en \mathcal{I} . Para ello, en cada iteración del bucle, se obtiene el conjunto de acciones relevantes, que son las que consiguen algún objetivo de \mathcal{G} . Si el conjunto *Relevante* resulta vacío entonces es que no existen acciones para conseguir los objetivos de \mathcal{G} , y el algoritmo devuelve fallo. En caso contrario, se selecciona aleatoriamente una acción y se añade al plan Π . El último paso actualiza el conjunto

\mathcal{G} eliminando los efectos ya conseguidos por la acción seleccionada y añadiendo al nuevo conjunto las precondiciones de la misma.

Algoritmo 13.1 Esquema básico de la búsqueda hacia atrás.

```

1:  $\Pi \leftarrow$  Plan vacío
2: bucle
3:   si  $\mathcal{G} \in \mathcal{I}$  entonces
4:     Devuelve  $\Pi$  {Éxito}
5:   fin si
6:    $Relevante \leftarrow \{a \mid a \text{ es una acción relevante para } \mathcal{G}\}$ 
7:   si  $Relevante = \emptyset$  entonces
8:     Devuelve fallo {No existe plan}
9:   fin si
10:  Seleccionar no determinísticamente y extraer  $a \in Relevante$ 
11:   $\Pi \leftarrow a \cdot \Pi$ 
12:   $\mathcal{G} = \mathcal{G} - Efe^+(a) \cup Pre(a)$ 
13: fin bucle

```

Los nodos del árbol que se generan en una búsqueda hacia atrás representan el conjunto de literales que restan por satisfacer para obtener una solución al problema, es decir, no representan estados del problema sino *estados objetivo*. Estableciendo una equivalencia con el Algoritmo 13.1, los nodos del árbol representarían los diferentes conjuntos \mathcal{G} que se generan en las distintas iteraciones de la búsqueda.

El proceso de expansión de un árbol en una búsqueda hacia atrás es muy similar al de la búsqueda hacia delante. La búsqueda de acciones relevantes durante la expansión de un estado objetivo (paso 6 del Algoritmo 13.1) supone la generación de sus nodos predecesores que a su vez se introducirán en la lista *ABIERTA* del árbol. Al igual que con la búsqueda hacia delante, la aplicación de una estrategia de búsqueda concreta determinaría el orden en el que los nodos son expandidos; sin embargo, en nuestro caso no utilizaremos ninguna estrategia en particular y en su lugar seleccionaremos no determinísticamente el nodo a expandir (paso 10 del Algoritmo 13.1).

Ejemplo 13.2. La Figura 13.2 muestra el árbol de búsqueda que se genera para el problema de transporte de ejemplo. El número asociado a cada nodo indica el orden de expansión del mismo. El primer estado objetivo del problema está constituido por el único literal del objetivo del problema, $\mathcal{G} = \{\text{pos}(p1, cB)\}$ y ése será el primer nodo a expandir. La única acción relevante para dicho literal es $d cg(p1, c1, cB)$, por lo que el nuevo estado objetivo (nodo 2) se compone de las dos precondiciones de dicha acción. Para obtener los siguientes estados predecesores se aplican todas las posibles acciones relevantes en el nodo 2, es decir acciones que consigan alguno de los dos literales de dicho nodo, y se actualiza el conjunto \mathcal{G} (tal y como se muestra en el último paso del Algoritmo 13.1). En el árbol de la Figura se pueden observar las siguientes situaciones:

1. La búsqueda desde el nodo 2 cuando se aplica la acción $cg(p1, c1, cB)$ generaría un estado repetido ya que se trata de la acción inversa que ha dado lugar al nodo 2 y por tanto se generaría un nodo predecesor que contendría como objetivos a conseguir el literal $\text{pos}(p1, cB)$, que es el mismo literal del nodo inicial \mathcal{G} . En

este caso se produciría un bucle al intentar satisfacer nuevamente el objetivo \mathcal{G} . Esta misma situación se produce al intentar satisfacer el literal $\text{pos}(p1, cA)$ del nodo 3, ya que la acción que consigue dicho literal es la inversa de la acción $\text{cg}(p1, c1, cA)$.

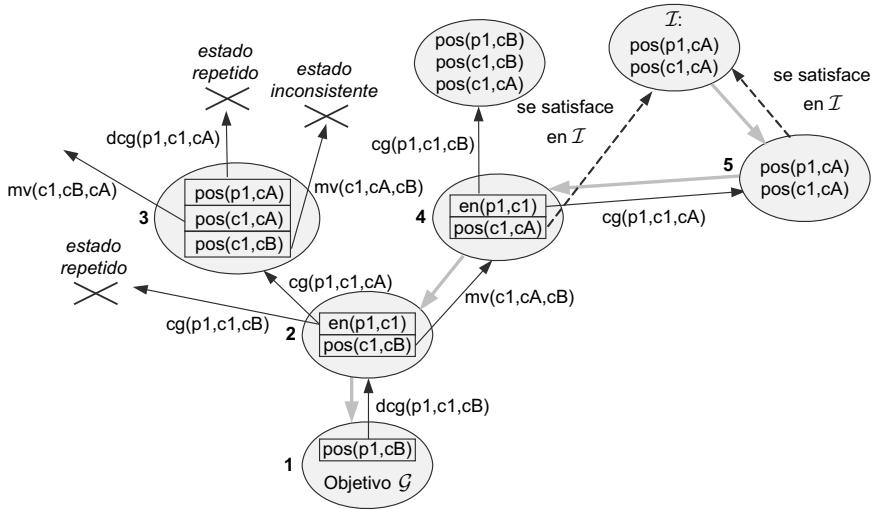


Figura 13.2: Ejemplo del árbol de búsqueda hacia atrás que se generara para el problema de transporte (véase la Tabla 13.1). En trazo gris se muestra la rama que constituye el plan.

2. Puede observarse cómo aparecen varios nodos marcados como *estado inconsistente*. Tomemos como ejemplo el literal $\text{pos}(c1, cA)$ del nodo 3. La acción que satisface dicho literal es $\text{mv}(c1, cB, cA)$, la cual contiene entre sus efectos negativos el literal $\text{pos}(c1, cB)$ que a su vez es un literal a satisfacer en el nodo 3; en este caso, se dice que el estado que se alcanzaría es inconsistente porque la acción que consigue un literal borra otro literal del conjunto \mathcal{G} . Además de que las acciones alcancen algunos literales de \mathcal{G} (acciones relevantes), las acciones no deben deshacer ningún literal de ese conjunto. Una acción que satisfaga esta propiedad se dice que es una acción consistente. Por tanto, la aplicación de una acción relevante a lleva a un estado inconsistente si se cumple $(\exists g \in \mathcal{G} \mid g \in Efe^-(a))$. Esta misma situación se produciría al intentar satisfacer los literales $\text{pos}(c1, cB)$ y $\text{pos}(c1, cA)$ del estado que se genera a partir del nodo 4 con la aplicación de la acción $\text{cg}(p1, c1, cB)$.
3. Una de las ramas que se generan en la expansión del nodo 4 lleva a un estado objetivo (nodo 5) cuyos literales se satisfacen en el estado inicial. El algoritmo devolvería, por tanto, el plan que se indica con las líneas de trazo gris, que es

el plan que se ha formado concatenando sucesivamente las acciones relevantes encontradas.

Nótese que en caso de generar un estado repetido o un estado inconsistente el algoritmo continuaría la expansión por cualquiera de los nodos de la lista *ABIERTA* del árbol (selección arbitraria). También debe observarse que, dado que se trata de una búsqueda regresiva, el orden en el que se encuentran las acciones del plan es el orden inverso en el que se encuentran las acciones en un proceso de búsqueda hacia delante; esto es, la primera acción encontrada será la última acción del plan.

Al igual que en la búsqueda hacia delante, un planificador regresivo genera un plan añadiendo pasos (acciones) totalmente instanciados al final del plan y se devuelve una solución totalmente ordenada cuando se alcanza el objetivo del problema. Por tanto, los planificadores regresivos son también planificadores de **orden total**. Sin embargo, una importante ventaja que presenta la búsqueda regresiva frente a la búsqueda progresiva es que permite considerar sólo acciones relevantes para la consecución de los objetivos, reduciendo de este modo el factor de ramificación del árbol de búsqueda en varios órdenes de magnitud. Por ejemplo, si existen tres paquetes y el objetivo es transportar sólo uno de ellos, las acciones que involucran a los otros dos no se considerarán. Es por esto que, en general, la búsqueda regresiva es más eficiente que la búsqueda progresiva para la resolución de problemas de planificación.

13.4.2.1 El algoritmo STRIPS

El algoritmo de planificación del sistema **STRIPS** está basado en un proceso de búsqueda hacia atrás. Se trata, en realidad, de un proceso recursivo que difiere del esquema general del Algoritmo 13.1 en los siguientes aspectos:

- En cada llamada recursiva del algoritmo **STRIPS**, los estados objetivos predecesores se forman únicamente con las precondiciones de la última acción, permitiendo así reducir sustancialmente el factor de ramificación.
- Si en el estado actual del problema se satisfacen todas las precondiciones de una acción, **STRIPS** ejecuta dicha acción en el estado actual del problema y actualiza el estado de planificación. Dado que la ejecución de una acción es una decisión irrevocable, puede suceder que alguno de los objetivos ya conseguidos se eliminen como consecuencia de dicha operación, es decir, al ejecutar una acción para conseguir un objetivo g_i se puede deshacer otro ya conseguido g_j . En este caso, se introduce de nuevo g_j como objetivo, o bien se realiza una vuelta atrás para explorar otro camino que no deshaga g_j . Con este tipo de funcionamiento se consigue igualmente reducir una gran parte del espacio de búsqueda.

13.5 Planificación de orden parcial

En la planificación de orden total las acciones del plan se obtienen en el mismo orden en que éstas se ejecutarían; esto es, el orden de planificación coincide con el

orden de ejecución. De este modo, si un planificador selecciona una acción *equivocada*, deberá incorporar una acción adicional para *deshacer* los efectos de la acción errónea y volver a un estado de planificación correcto. En problemas complejos se suceden múltiples interacciones entre los objetivos del problema porque las acciones que se utilizan para resolver un objetivo pueden interferir en la solución de otro objetivo. Un planificador que tenga en cuenta esta característica generará un plan entrelazado en el que se trabaja simultáneamente con múltiples objetivos del problema. De este modo, el planificador tendrá que tomar decisiones sobre cómo las acciones que resuelven una parte del problema afectan a la resolución del resto del problema. A este tipo de aproximación se le denomina **planificación de orden parcial** (POP).

La ventaja de la aproximación POP es la flexibilidad a la hora de establecer un orden entre las acciones que componen un plan, ya que los planificadores pueden primero tomar decisiones sobre las partes importantes del plan, en lugar de forzar un orden cronológico entre las acciones del mismo. En POP se trabaja sobre los objetivos del problema simultáneamente y se mantiene un orden parcial entre las acciones sin tener que comprometer un orden concreto entre las mismas, hasta que la propia estructura del plan determina cuál debe ser este orden. A la estrategia general de retrasar una decisión durante el proceso de búsqueda se le conoce como **menor compromiso**. Por ejemplo, si se dispone de dos paquetes $\{p_1, p_2\}$ en la ciudad c_A que deben ser cargados en el camión c_1 , el orden en el que se realice las acciones $cg(p_1, c_1, c_A)$ y $cg(p_2, c_1, c_A)$ no es relevante para la consecución del objetivo del problema. Un POP deja abierto el orden entre ambas acciones, es decir, establece un orden parcial entre las mismas hasta que las condiciones del problema determinen cuál de los dos paquetes debe cargarse antes. Si la estructura o condiciones del problema no impone un orden explícito entre los dos paquetes, entonces un POP puede devolver cualquier orden entre las dos acciones (cargar p_1 primero y luego p_2 , o viceversa) o bien devolver las acciones en paralelo si el agente de ejecución es capaz de ejecutar acciones concurrentemente.

13.5.1 Estructura de un plan de orden parcial

La Figura 13.3 muestra un plan de orden parcial para el problema de transporte de ejemplo, correspondiente al problema de la Tabla 13.1, en tres etapas a lo largo de su construcción. Los elementos que definen un plan de orden parcial son los siguientes:

- **Pasos del plan.** Un paso es una versión total o parcialmente instanciada de un operador del problema. Por consiguiente, cada paso de un plan se compone de un conjunto de precondiciones y efectos que están, a priori, total o parcialmente instanciados. Cuando finalmente se obtiene el plan que resuelve el problema, los pasos del plan serán instancias completas de los operadores del plan (acciones). En la Figura 13.3-a) se muestra el plan vacío inicial, el cual se compone de dos pasos ficticios, el paso inicial I y el paso final F. El paso I no tiene precondiciones y sus efectos se corresponden con los literales del estado inicial. El paso F tiene como precondiciones el objetivo del problema, en este caso $\{\text{pos}(p_1, c_B)\}$, y no tiene efectos. En la Figura 13.3-b) se puede observar que aparecen tres

nuevos pasos: los pasos $\text{dcg}(p1, c1, cB)$ y $\text{mv}(c1, cA, cB)$ se corresponden con instanciaciones totales de los operadores descargar y mover, respectivamente, mientras que el paso $\text{cg}(p1, c1, ?ciu)$ es una versión parcialmente instanciada del operador cargar ya que el parámetro $?ciu$ podría tomar, a priori, los valores cA o cB .

- **Restricciones de orden.** Una restricción de orden (**precedencia**) entre dos pasos del plan P_i , P_j se representa como $P_i < P_j$ y denota que el paso P_i debe ejecutarse antes que el paso P_j , aunque no necesariamente inmediatamente antes. Las restricciones de orden establecen una relación de orden parcial entre sus elementos. Esta relación de orden parcial es una relación binaria irreflexiva, antisimétrica (una restricción de orden no se puede añadir si genera un ciclo) y transitiva. En la Figura 13.3-a) se establece una restricción de orden entre los pasos I y F para indicar que el paso I siempre se ejecutará antes que el paso F, mientras que en la Figura 13.3-c) se establece una restricción de orden entre los pasos $\text{cg}(p1, c1, ?ciu)$ y $\text{mv}(c1, cA, cB)$ para indicar la precedencia de ejecución del primer paso.
- **Restricciones entre variables.** Un POP también puede contener un conjunto de restricciones sobre variables como restricciones de desigualdad del tipo $?x \neq ?y$, donde $?x$ es una variable de un paso del plan e $?y$ es una constante u otra variable. Otro tipo de restricciones que puede contener son las denominadas restricciones de *codesignación* o *unificación*. Éstas son restricciones del tipo $?x = ?y$ donde se fuerza que la variable $?x$ tenga el mismo valor que $?y$. En la Figura 13.3-c) se establece la restricción de unificación $?ciu=cA$ del paso $\text{cg}(p1, c1, ?ciu)$. De este modo, las precondiciones del paso quedarán totalmente instanciadas y se podrán establecer sendos enlaces causales con los efectos del paso I. Nótese que el hecho de no ligar un valor a una variable en el momento de insertar el paso es otra variante de menor compromiso. Si en el momento de insertar un paso se desconoce el valor exacto de uno de sus parámetros, éste puede dejarse sin instanciar y en su defecto se le asigna todos los posibles valores del dominio de la variable. La restricción de unificación para dicha variable se establecerá en el momento que se disponga de suficiente información en el plan para poder asignar un valor irrevocablemente. La ventaja de aplicar una estrategia de menor compromiso en las restricciones sobre variables es que, de ese modo, se evita generar tantos planes parciales como valores pueda tomar la variable, y en su lugar se genera un único plan parcial donde la variable contiene inicialmente todos los posibles valores de su dominio.
- **Enlaces causales y amenazas.** Un enlace causal entre dos pasos de un plan P_i , P_j se escribe como $\langle P_i, l, P_j \rangle$ y denota que P_i tiene un efecto que consigue el literal l para una precondición de P_j . En las Figuras 13.3-b),c) se pueden encontrar varios enlaces causales; por ejemplo, $\langle \text{dcg}(p1, c1, cB), \text{pos}(p1, cB), F \rangle$ o bien $\langle \text{mv}(p1, cA, cB), \text{pos}(c1, cB), \text{dcg}(p1, c1, cB) \rangle$. Todo enlace causal establece por defecto una restricción de orden entre los dos pasos. En un enlace causal el literal que el paso productor consigue para el paso consumidor debe mantenerse

cierto desde el punto de tiempo del primer paso hasta el punto de tiempo del segundo. Es decir, no debe existir acción alguna que entre en conflicto con el enlace causal. Un paso P_k entra en conflicto o **amenaza** un enlace causal $\langle P_i, l, P_j \rangle$ si P_k tiene un efecto $\neg l$ y P_k puede situarse entre P_i y P_j . Para resolver la amenaza el paso P_k debería situarse antes de P_i (método *promoción*) o después de P_j (método *democión*). En la Figura 13.3-c), una vez establecida la restricción de unificación del parámetro $?ciu$, el paso $mv(c1, cA, cB)$ amenaza el enlace causal $\langle I, pos(c1, cA), cg(p1, c1, cA) \rangle$ porque dicho paso borra el efecto $pos(c1, cA)$. Como el paso $mv(c1, cA, cB)$ no puede situarse antes del paso I , la única posibilidad de resolver la amenaza es aplicar democión para que el paso $mv(c1, cA, cB)$ se sitúe detrás de $cg(p1, c1, cA)$. Es interesante destacar que se pueden generar amenazas *potenciales* cuando los literales en conflicto involucran variables, en cuyo caso se pueden resolver a través de restricciones de desigualdad. Supongamos que el parámetro $?ciu$ del paso $cg(p1, c1, ?ciu)$ no se ha instanciado aún. Dado que el paso $mv(c1, cA, cB)$ tiene el efecto $\neg pos(c1, cA)$ se produce una amenaza potencial porque los valores de la variable $?ciu$ serían todos los posibles valores de su dominio, es decir las ciudades $\{cA, cB\}$. Una posible forma de resolver una potencial amenaza es eliminar del dominio de la variable el valor que provoque la amenaza, es decir, establecer la restricción $?ciu \neq cA$ (método *separación*).

13.5.2 Búsqueda en un espacio de planes para POP

Un proceso de búsqueda en un espacio de planes realiza una exploración en un árbol donde cada nodo representa un plan parcial. El Algoritmo 13.2 muestra el esquema básico de búsqueda en un espacio de planes. El nodo raíz del árbol será un plan vacío consistente únicamente en los pasos I y F . Dicho plan se introduce en la lista *Lista_planes*, donde se irán añadiendo sucesivamente los nodos frontera del árbol y de donde se escogerán aleatoriamente los nodos, es decir los planes Π (inicialmente se escogerá el plan vacío ya que es el único plan de la lista). Una vez seleccionado el plan Π con el que se va a trabajar en la actual iteración del algoritmo, se obtienen las precondiciones no resueltas y amenazas del plan y se introducen en la lista *Pendiente*. Si no hay nada pendiente de resolución entonces eso indica que el plan es correcto (resuelve el problema) y se devuelve dicho plan. En caso contrario, se selecciona uno de los problemas pendientes de resolución, se extrae de la lista y se buscan todas las formas de resolver dicho problema. Si se trata de una precondición no resuelta de un paso, existen dos formas genéricas de resolución:

- **Reutilizar un paso existente.** Por cada paso existente en el plan que resuelve dicha precondición se genera un nodo sucesor o plan parcial que incluye el enlace causal correspondiente.
- **Insertar un nuevo paso.** Por cada acción relevante para la precondición se genera un nodo sucesor o plan parcial, incluyendo un nuevo paso asociado a la acción y el enlace causal correspondiente.

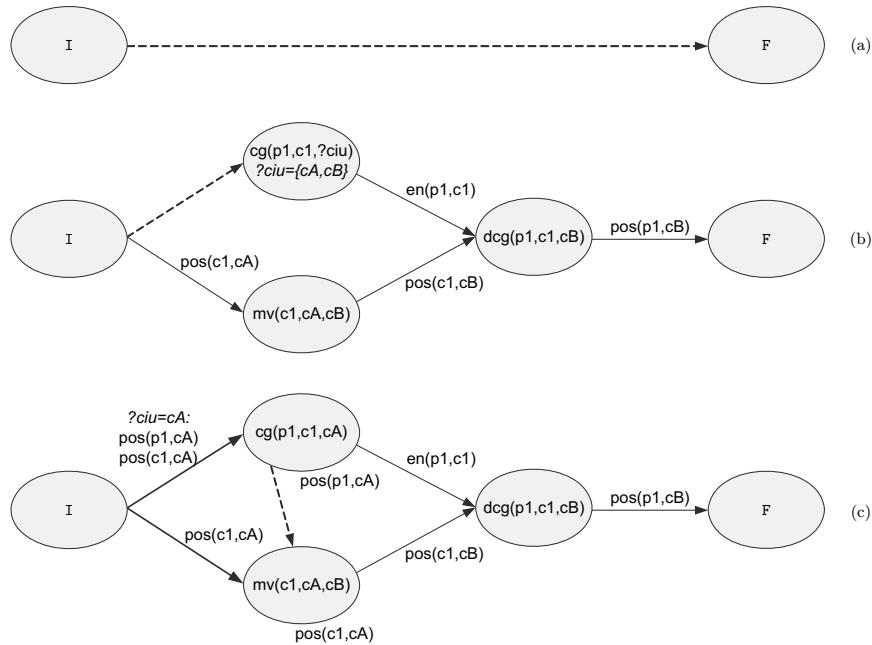


Figura 13.3: Ejemplo de un plan de orden parcial para el problema de transporte (véase la Tabla 13.1). Las flechas continuas representan enlaces causales y las punteadas restricciones de orden.

Si el problema pendiente de resolución es una amenaza, entonces se genera un nodo sucesor o plan parcial por cada uno de los métodos de resolución (promoción, democión o separación) que sean aplicables. Por ejemplo, en la amenaza de la Figura 13.3-c), no es posible aplicar el método promoción porque el paso $mv(c1,cA,cB)$ no puede situarse delante del paso **I**.

A tenor de lo explicado se puede ver que las acciones aplicables en esta búsqueda no son acciones ejecutables en el mundo real sino acciones sobre los planes, como añadir un paso, establecer un orden entre acciones, etc. El algoritmo continúa seleccionando el siguiente plan parcial a expandir y repitiendo las mismas operaciones. El proceso termina cuando: 1) se selecciona un plan parcial que es correcto, es decir, que resuelve el problema, o 2) la lista de nodos a expandir del árbol se queda vacía, lo que indica que el problema no tiene solución. En el caso de nuestro ejemplo se seleccionaría un nodo con el plan parcial de la Figura 13.3-c), que contiene justamente el plan que resuelve el problema.

Un POP que utilice el algoritmo de la Figura 13.2 generará todas las posibles formas de resolver tanto las precondiciones pendientes como las amenazas existentes. Por otro lado, la selección que se realiza en el paso 8 del algoritmo es un paso

determinístico, ya que se deben resolver **todas** las precondiciones y amenazas antes de alcanzar un plan solución. Esto indica que un algoritmo POP generará todos los posibles planes parciales para resolver un problema. Sin embargo, el procedimiento POP será completo sólo si garantiza que se explorarán todos los planes parciales hasta un determinado nivel de profundidad del árbol. Para ello sería necesario emplear una estrategia de búsqueda de profundización iterativa, como las mostradas en la sección 8.4.1.4 del capítulo 8, donde se va incrementando progresivamente el límite de la solución buscada. En caso contrario, la búsqueda podría seleccionar un camino del espacio de búsqueda y continuar la exploración de dicho camino a niveles cada vez más profundos, añadiendo indefinidamente nuevas acciones al plan actual sin realizar ninguna vuelta atrás.

Algoritmo 13.2 Esquema básico de búsqueda en un espacio de planes para POP

```

1: Lista_planes  $\leftarrow \{\text{Plan vacío}\}$ 
2: repetir
3:   Seleccionar no determinísticamente y extraer  $\Pi \in \text{Lista_planes}$ 
4:   Pendiente  $\leftarrow \text{prec\_no\_resueltas}(\Pi) \cup \text{amenazas}(\Pi)$ 
5:   si Pendiente =  $\emptyset$  entonces
6:     Devuelve  $\Pi$  {Éxito}
7:   fin si
8:   Seleccionar y extraer  $\Phi \in \text{Pendiente}$ 
9:   Relevante  $\leftarrow \{\Pi_r\} \forall \Pi_r \text{ que resuelve } \Phi$  {Cada  $r$  es una forma (plan parcial) de resolver  $\Phi$ }
10:  si Relevante  $\neq \emptyset$  entonces
11:    Lista_planes  $\leftarrow \text{Lista_planes} \cup \text{Relevante}$ 
12:  fin si
13: hasta lista_planes =  $\emptyset$ 
14: Devuelve fallo {No existe plan}

```

A pesar de las ventajas que las aproximaciones POP aportan sobre la búsqueda basada en estados, la complejidad de la resolución de un problema sigue siendo muy elevada. El factor de ramificación de un árbol de planes se puede expresar con la siguiente fórmula: $(B_e + B_n) * m^a$, donde B_e y B_n son las formas de resolver una precondición con pasos existentes en el plan o introduciendo pasos nuevos, respectivamente; m son las formas de resolver una amenaza y a es el número de amenazas de un plan. Hay que tener en cuenta que por cada alternativa de resolver una precondición se genera un plan parcial, y para cada uno de ellos hay que estudiar y resolver las amenazas que puedan existir (m^a). Considerando que un plan contiene p precondiciones, tendríamos que el factor efectivo de ramificación sería $((B_e + B_n) * m^a)^p$, y asumiendo que el plan solución para un problema tiene n pasos, tendríamos que el número de nodos que se pueden generar en un proceso de búsqueda en un espacio de planes es $((B_e + B_n) * m^a)^{pn}$. En términos efectivos este factor puede resultar en una explosión combinatoria para resolver instancias de problemas de una cierta complejidad. Aunque es innegable el avance que supusieron las aproximaciones POP a la planificación, éstas resultan aún demasiado costosas.

Ejemplo 13.3. Retomando el ejemplo de la Figura 13.3, veremos cómo se desarrolla el proceso de búsqueda del Algoritmo 13.2 para transformar el plan de la Figura 13.3-a) en el plan de la Figura 13.3-c). Inicialmente, el plan vacío se corresponde con

el que aparece en la Figura 13.3-a). La única precondición pendiente de resolución es $\text{pos}(p1, cB)$ del paso F por lo que la lista Pendiente sólo contendrá este objetivo. Para resolver dicha precondición sólo existe la alternativa de añadir el paso $\text{drg}(p1, c1, cB)$ que proporciona el enlace causal correspondiente; por consiguiente, sólo se generará un nodo sucesor que contendrá el plan parcial donde se incluye el nuevo paso para descargar el paquete. En la siguiente iteración se escoge el único plan de Lista_Planes, y se analiza los dos objetivos de la lista Pendiente, que serán las dos precondiciones $\{\text{en}(c1, p1), \text{pos}(c1, cB)\}$ del paso $\text{drg}(p1, c1, cB)$. Supongamos que se escoge la precondición $\text{en}(p1, c1)$. Un análisis de la resolución de dicha precondición revela los siguientes aspectos:

- No existe forma de resolver dicha precondición mediante la reutilización de un paso existente, ya que el único paso anterior a $\text{drg}(p1, c1, cB)$ es I, y éste no contiene un efecto que pueda resolver la precondición.
- Una forma de resolver la precondición es añadir el paso $\text{cg}(p1, c1, cA)$ o bien el paso $\text{cg}(p1, c1, cB)$. En principio esta doble posibilidad generaría dos nodos sucesores pero, dado que se trata del mismo operador cuyo tercer argumento puede asumir dos valores diferentes, se genera un único plan donde la variable $?ciu$ toma provisionalmente todos los valores de su dominio, $?ciu=\{cA, cB\}$.

De nuevo se tiene que la lista Lista_Planes contiene un único plan, el cual consiste momentáneamente en los pasos $\{I, \text{cg}(p1, c1, ?ciu), \text{drg}(p1, c1, cB), F\}$. La lista Pendiente estaría ahora formada por la precondición pendiente del paso descargar y las dos precondiciones del paso cargar. Asumiendo que se selecciona $\text{pos}(c1, cB)$ del paso descargar, la única posibilidad de resolverla es insertando un nuevo paso, $\text{mv}(c1, cA, cB)$. En este punto existen tres precondiciones pendientes de resolución, la del paso $\text{mv}(c1, cA, cB)$ y las dos del paso $\text{cg}(p1, c1, ?ciu)$. Si se selecciona la precondición $\text{pos}(c1, cA)$ del paso mover, se puede reutilizar el paso existente I y establecer el enlace $\langle I, \text{pos}(c1, cA), \text{mv}(c1, cA, cB) \rangle$ o bien insertar un nuevo paso $\text{mv}(c1, cB, cA)$ cuyo objetivo sería llevar el camión a cA. En este punto del árbol se generaría una ramificación con las dos posibles soluciones.

Supongamos que el algoritmo escoge el plan donde se establece el enlace causal con el paso I, situación que se corresponde con la Figura 13.3-b). En este punto sólo quedarían las dos precondiciones del paso $\text{cg}(p1, c1, ?ciu)$ en la lista Pendiente. Para resolver la precondición $\text{pos}(p1, ?ciu)$ existen dos opciones, al igual que con la precondición del paso mover: establecer un enlace causal con el paso existente I, lo que ligaría la variable $?ciu$ al valor cA, o bien añadir un nuevo paso $\text{drg}(p1, c1, ?ciu)$. Asumiendo que el proceso de búsqueda escoge el plan donde se reutiliza el paso I, la resolución de la siguiente precondición $\text{pos}(c1, cA)$ produciría las mismas alternativas que anteriormente con la precondición del paso mover. Si el algoritmo de búsqueda escoge el plan que reutiliza el paso I, todas las precondiciones estarán resueltas pero entonces aparecerá la amenaza del paso $\text{mv}(c1, cA, cB)$ al enlace causal $\langle I, \text{pos}(c1, cA), \text{cg}(p1, c1, cA) \rangle$. De los dos métodos posibles para resolver la amenaza, promoción y democión, sólo es aplicable el segundo ya que ningún paso puede situarse antes del paso I. Se establece por tanto la restricción de orden que aparece en la

Figura 13.3-c), la cual determina que la operación de cargar debe realizarse antes que mover el camión.

En la siguiente iteración, el algoritmo puede seleccionar el nodo que representa el plan de la Figura 13.3-c) o bien cualquiera de los nodos abiertos correspondientes a las ramificaciones producidas durante el proceso de búsqueda. Si se selecciona el nodo de la Figura 13.3-c), como no existen precondiciones ni amenazas que resolver, el algoritmo termina con éxito devolviendo dicho plan.

13.5.3 Heurísticas para planificación de orden parcial

Los dos puntos claves en una búsqueda en un espacio de planes son la selección del plan de *Lista_Plans* y la selección, dentro de dicho plan, del objetivo pendiente a estudiar. Estos dos puntos son críticos, ya que de las elecciones tomadas dependerá el coste de resolución del problema. Existen diferentes estrategias o heurísticas enfocadas a tomar la mejor decisión posible y enfocar así la búsqueda hacia el camino donde se encuentra la solución.

13.5.3.1 Heurísticas para selección de planes

La estrategia habitual es ordenar los nodos frontera del árbol por el coste total de los operadores del plan (número de pasos) y seleccionar el plan con el mínimo coste. Se puede diseñar un algoritmo A^* con función $f(\Pi) = g(\Pi) + h(\Pi)$, donde $g(\Pi)$ representa el coste de la solución desde el nodo raíz hasta plan Π , y $h(\Pi)$ representa la estimación del coste de la solución desde el plan Π hasta el objetivo. Para diseñar una buena estrategia heurística hay que determinar cómo afectan cada uno de los componentes de un plan a los factores $g(\Pi)$ y $h(\Pi)$:

- **Número de pasos N** : se utiliza habitualmente como una medida del coste del camino ya que la complejidad de un plan viene dado por el número de pasos que contiene.
- **Precondiciones sin resolver OP** : dado que para cada precondición pendiente de resolver se debe introducir un paso, OP se puede incluir como medida heurística en $h(\Pi)$. En principio, podría deducirse que OP sobreestima $h^*(\Pi)$ ya que no se tiene en cuenta la reutilización de pasos existentes para resolver una precondición, es decir, no siempre es necesario añadir un paso para resolver una precondición. Sin embargo, la posibilidad de que OP sobreestime el coste restante del plan no es habitual porque, típicamente, se necesitarán pasos futuros adicionales para conseguir algunos objetivos y estos pasos, a su vez, requerirán nuevos pasos que no se contemplan en el valor devuelto por OP . Por tanto, OP se puede incluir en $h(\Pi)$ con una alta garantía de que en la mayoría de situaciones no sobreestimará $h^*(\Pi)$.
- **Amenazas A** : son un factor circunstancial de un plan que aparecen en un momento concreto y que pueden resolverse, o bien desaparecer, a medida que el plan se va construyendo. Por dicha razón, A no se incluye en $h(\Pi)$ pero puede

considerarse como una medida del coste de encontrar una solución (a mayor número de amenazas más costoso será, en principio, encontrar la solución).

La combinación que se ha utilizado habitualmente en las aproximaciones POP es $f(\Pi) = N(\Pi) + OP(\Pi)$, donde existen altas garantías de que $OP(\Pi) < h^*(\Pi)$ y por tanto de que se pueda encontrar el plan óptimo para el problema, es decir el plan de menor número de pasos o acciones.

13.5.3.2 Heurísticas para selección de objetivo pendiente

Existen múltiples estrategias que se han aplicado en las aproximaciones POP. Básicamente, las estrategias existentes priorizan entre escoger una amenaza o una precondición pendiente de resolución y entre ellas se establece otra prioridad para determinar el objetivo candidato a ser estudiado:

- **LIFO.** Esta estrategia aplica una estrategia LIFO para seleccionar el siguiente objetivo pendiente a estudiar.
- **Retrasar las amenazas potenciales.** Esta estrategia determina que es preferible ignorar cualquier amenaza potencial que pueda haber en un plan y que los objetivos pendientes deben resolverse en el siguiente orden: 1) amenazas no resolubles, 2) amenazas que puedan resolverse con un solo método, 3) precondiciones sin resolver, y 4) amenazas que puedan resolverse con más de un método.
- **Objetivo de menor coste.** Esta estrategia selecciona la amenaza o precondición pendiente de menor coste de reparación, es decir, aquel objetivo pendiente con menos alternativas de solución. En el caso de precondiciones, el número de alternativas vendrá dado por el número de pasos existentes (B_e) y pasos nuevos (B_n) que pueden crearse para resolver la precondición.
- **ZLIFO (Zero LIFO).** El objetivo de esta estrategia es dar mayor prioridad a aquellas precondiciones que puedan resolverse determinísticamente (0 ó 1 solución). Para ello, establece el siguiente nivel de prioridades: 1) escoger una amenaza (si hay varias, aplicar una estrategia LIFO), 2) escoger una precondición pendiente con 0 formas de resolverse (precondición irresoluble), 3) escoger una precondición para la que sólo existe una alternativa de solución, y 4) escoger una precondición pendiente aplicando una estrategia LIFO. A diferencia de la heurística objetivo de menor coste, ZLIFO no calcula el coste de reparación de las precondiciones ni de las amenazas, sino que simplemente determina si existen 0, 1 o varias formas de resolverlas. En el último caso se selecciona una precondición aplicando la estrategia LIFO.

Aunque no existe una estrategia heurística universal para toda la tipología de problemas, la estrategia ZLIFO ha sido la heurística que mejores resultados ha obtenido en términos generales.

Ejemplo 13.4. Consideremos de nuevo el ejemplo utilizado para mostrar el proceso de búsqueda en el espacio de planes de la Figura 13.3, situándose en el punto de resolución de la precondición $\text{pos}(c1, cA)$ del paso $\text{mv}(c1, cA, cB)$. Existen dos alternativas: reutilizar el paso I, o introducir un nuevo paso $\text{mv}(c1, cB, cA)$. La aplicación de la función $f(\Pi) = N(\Pi) + OP(\Pi)$ para cada una de estas dos alternativas daría los siguientes resultados:

- **Reutilizar paso I.** El número de pasos del plan si se reutiliza el paso I sería 5, y el número de precondiciones por resolver sería 2 (las dos precondiciones del paso cargar). Por tanto: $f(\Pi) = N(\Pi) + OP(\Pi) = 5 + 2 = 7$.
- **Insertar paso nuevo.** En este caso el número de pasos del plan sería 6 y el número de precondiciones por resolver sería 3 (las dos precondiciones del paso cargar y la precondición del paso $\text{mv}(c1, cB, cA)$). Por tanto: $f(\Pi) = N(\Pi) + OP(\Pi) = 6 + 3 = 9$.

De acuerdo a esta información, la alternativa seleccionada sería la de reutilizar el paso I pues es menos costosa que la de insertar un paso nuevo.

13.6 Planificación basada en grafos de planificación

La planificación basada en grafos de planificación consiste en la construcción, análisis y explotación de una estructura bidimensional, que representa de forma compacta las proposiciones, acciones y ciertas restricciones entre ambas que aparecen durante la resolución de un problema de planificación.

13.6.1 Grafos de planificación

Un grafo de planificación es una estructura de tamaño polinómico, similar a las empleadas en programación dinámica, que modela la parte estática (proposiciones y acciones) y dinámica (relaciones de orden). La única restricción que impone el grafo de planificación es la de manejar un enfoque **proposicional** (véase la Asunción A4 en la sección 13.1) basado en STRIPS, es decir con literales positivos. Tanto las acciones como proposiciones (literales positivos) deben estar totalmente instanciadas, esto es sin variables. Más precisamente, un grafo de planificación es un grafo dirigido multivel con dos tipos de nodos (proposición y acción) y tres tipos de aristas (aristas-Pre, aristas-Efe⁺ y aristas-Efe⁻). Cada nivel t del grafo contiene, a su vez, un nivel de acción $A_{[t]}$ y uno de proposición $P_{[t]}$ con las acciones y proposiciones presentes, respectivamente, en dicho nivel.

Ejemplo 13.5. Si consideramos el problema de transporte definido en la Tabla 13.1, el grafo de planificación resultante es el que se muestra en la Figura 13.4. Este grafo se construye incrementalmente a partir de la información de la situación inicial I , que constituye el nivel $P_{[0]}$. Las aristas del grafo sólo pueden unir nodos de niveles adyacentes; una acción del nivel t está conectada a todas sus precondiciones en el nivel $t - 1$ mediante aristas-Pre y a sus efectos en el mismo nivel t , tanto positivos

como negativos mediante aristas-Efe⁺ y aristas-Efe⁻, respectivamente. Así por ejemplo, la acción $cg(p1, c1, cA)$ en $A_{[1]}$ tiene como precondiciones en $P_{[0]}$ el conjunto $\{pos(c1, cA), pos(p1, cA)\}$, como efecto positivo $en(p1, c1)$ y negativo $pos(p1, cA)$, ambos en $P_{[1]}$. Como se puede observar, en los niveles de acción existen unas acciones artificiales denominadas *no-op* (*no-operación*). Estas acciones son habituales en los grafos de planificación y, aunque no aportan nueva información al problema (su único efecto positivo es la precondición que requiere), permiten propagar información a lo largo del grafo, simplificando el problema marco y garantizando la persistencia de las proposiciones. En general, existen dos condiciones para detener la construcción del grafo de planificación: 1) cuando en un nivel de proposición se alcanzan todos los objetivos del problema, o 2) cuando se alcanza un nivel que es idéntico al anterior, tanto a nivel de proposiciones como de acciones. En el grafo de la Figura 13.4, la primera condición se cumple en $P_{[2]}$ al alcanzarse el objetivo $pos(p1, cB)$. La segunda condición se cumpliría tras generar el nivel 4 que coincidiría plenamente con el nivel 3 ($A_{[4]} = A_{[3]}$ y $P_{[4]} = P_{[3]}$), al haberse generado todas las posibles proposiciones y acciones (véase el ejercicio propuesto 6).

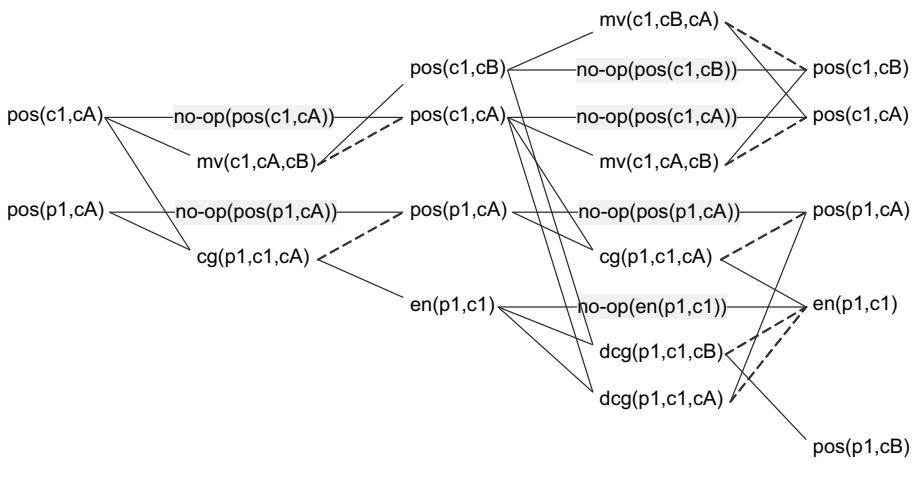


Figura 13.4: Ejemplo del grafo de planificación para el problema de transporte. Las aristas-Pre y aristas-Efe⁺ se representan mediante líneas continuas, mientras que las aristas-Efe⁻ se representan por líneas punteadas.

La construcción de un grafo de planificación no conlleva restricciones muy estrictas. En un nivel de acción $A_{[t]}$ aparecen todas las acciones (incluidas las acciones *no-op*) cuyas precondiciones están presentes en el nivel $P_{[t-1]}$, con independencia de si dichas precondiciones pueden coexistir simultáneamente, o si las acciones pueden ejecutarse concurrentemente en el mundo real. Análogamente, en un nivel de proposición $P_{[t]}$ aparecen todas las proposiciones que se han alcanzado a lo largo del grafo de

planificación, aunque haya acciones que las eliminen como parte de sus efectos Efe^- . Por lo tanto, es importante darse cuenta de que un nivel de un grafo de planificación no representa un estado individual válido, sino la unión de varios de ellos que no siempre pueden satisfacerse simultáneamente en el mundo real. Por esta razón, a un grafo de planificación que no contempla los efectos negativos (Efe^-) de las acciones se le conoce habitualmente como **grafo de planificación relajado**. Por ejemplo, en el nivel $P_{[1]}$ del grafo de la Figura 13.4 aparecen simultáneamente las proposiciones $\{\text{pos}(c1, cA), \text{pos}(c1, cB)\}$ como resultado de una relajación del problema original, a pesar de ser imposible para cualquier camión. Por el contrario, una información importante que se deduce de este grafo es que si una proposición o acción no aparece en un nivel t es porque no existe ninguna combinación factible que permita alcanzar dicha proposición o acción en t pasos de planificación. Por ejemplo, la acción $mv(c1, cB, cA)$ no aparece hasta el nivel $A_{[2]}$, lo que indica que no existe ninguna forma de ejecutar esa acción en el nivel $A_{[1]}$ (obviamente, para mover el camión $c1$ desde la ciudad cB primero hay que llegar a cB , lo que requiere al menos un paso de planificación). No obstante, una vez una acción o proposición aparece en un nivel, se mantiene en todos los niveles siguientes. Esto es consecuencia directa de la aplicación de las acciones **no-op**, que mantienen la proposición invariable a lo largo de los siguientes niveles, permitiendo la ejecución de las acciones que las utilizan como precondiciones. Por esta razón se dice que en un grafo de planificación el número de proposiciones y acciones es monótonamente creciente, pues nunca disminuye.

Los grafos de planificación cumplen varias propiedades que los hacen muy interesantes como fuente básica de información para estimar y mejorar el proceso de planificación:

1. Un grafo de planificación mantiene una noción interna del tiempo mediante los niveles o pasos de planificación, y representa implícitamente muchas de las restricciones entre acciones y proposiciones. Mediante las aristas-Pre y aristas- Efe^+ se pueden representar restricciones cualitativas entre acciones, de forma similar a como se hace en planificación de orden parcial con la utilización de enlaces causales.
2. Un grafo de planificación codifica un plan que se encuentra *a mitad camino* entre un plan de orden parcial y uno de orden total. En cada nivel del grafo las acciones se ejecutan concurrentemente (sin imponer un orden de ejecución, tal y como ocurre en un enfoque de orden parcial), pero los niveles deben ejecutarse de forma ordenada (tal y como ocurre en un enfoque de orden total). De esta forma, las acciones que se ejecutan en el nivel t se podrán ejecutar en cualquier orden, pero siempre antes que las del nivel $t + 1$.
3. Un grafo de planificación se construye con complejidad temporal y espacial polinómica con respecto al tamaño del problema. La prueba de esta afirmación es sencilla ya que en un dominio proposicional las acciones no pueden crear nuevas proposiciones. Esto obliga a que las acciones y proposiciones sean conjuntos finitos y que el número máximo de nodos acción/proposición en un nivel de acción/proposición también lo sea. En el ejemplo puede comprobarse cómo el

grafo no crece indefinidamente; por un lado, las proposiciones y acciones son fijas y constituyen un conjunto finito y, por el otro, los pasos de planificación también son finitos pues llega un punto en el que un nivel es idéntico al anterior.

4. Si existe un plan válido de t pasos para un problema de planificación, dicho plan existe como un subgrafo del grafo de planificación correspondiente de t niveles. Inversamente, si no existe ningún plan válido de t pasos para el problema, no existe ningún subgrafo en el grafo de planificación de t niveles que resuelva el problema.

13.6.2 Extracción de planes: Graphplan

Graphplan es, sin duda, el planificador basado en grafos de planificación más conocido. En lugar de comenzar directamente con un proceso de búsqueda hacia delante o hacia atrás desde la situación inicial o final, respectivamente, actúa en dos etapas. En la primera, Graphplan construye un grafo de planificación hasta un nivel t y, en la segunda, realiza una búsqueda regresiva para extraer un plan de longitud t de dicho grafo.

13.6.2.1 Relaciones de exclusión mutua

Un grafo de planificación relajado resulta demasiado permisivo en cuanto a las acciones o proposiciones que pueden aparecer en un determinado nivel, al representar la unión de muchos estados. Si observamos el grafo de la Figura 13.4 puede observarse cómo en el nivel $A_{[1]}$ podrían ejecutarse simultáneamente las acciones $cg(p1, c1, cA)$ y $mv(c1, cA, cB)$. Esto hace que en el nivel $P_{[1]}$ se disponga simultáneamente de las proposiciones $\{pos(c1, cB), en(p1, c1)\}$, junto con el efecto $pos(c1, cA)$, resultante de aplicar la acción $no-op(pos(c1, cA))$. Obviamente, ninguna de estas dos situaciones es factible en un plan válido. Por lo tanto, resulta necesario contar con un mecanismo que permita conocer cuáles son las relaciones de exclusión en un nivel, es decir, qué elementos (proposiciones/acciones) no pueden coexistir simultáneamente. Adicionalmente, supongamos un problema cuyo plan solución requiere tres pasos de planificación¹. En tal caso, comenzar la búsqueda desde el nivel 1 ó 2 conducirá inevitablemente al fracaso, y será necesario alcanzar el nivel 3 antes de comenzar la etapa de búsqueda. De nuevo, contar con un mecanismo de propagación de relaciones de exclusión puede evitar esta búsqueda infructuosa.

Graphplan amplía la construcción del grafo de planificación con un razonamiento sobre relaciones de exclusión mutua entre pares de nodos de un mismo nivel, bien sea acción-acción o proposición-proposición. En general, se dice que dos acciones o proposiciones de un mismo nivel son mutuamente excluyentes (*mutex*) si ningún plan válido puede contenerlas simultáneamente, es decir, no pueden coexistir en el mundo real. El cálculo de esta información de exclusión mutua produce un grafo de planificación más informado que permite detectar con mayor precisión el nivel desde el

¹El lector debe recordar que un grafo de planificación de tres pasos (niveles) de planificación se traducirá en un plan de, al menos, tres acciones (una por nivel).

que comenzar la búsqueda y restringir el espacio de búsqueda. En general, identificar todas las relaciones de exclusión puede resultar muy costoso, por lo que Graphplan calcula y propaga estas relaciones mediante la aplicación de unas reglas sencillas. En el caso de las acciones, dos acciones a y b son mutex cuando: 1) a borra una precondición o efecto positivo de b (*interferencia*), o 2) una precondición de a y una de b se han marcado como mutuamente excluyentes en el nivel de proposición anterior (*necesidades competitivas*). En el caso de las proposiciones, dos proposiciones p y q son mutex si todas las formas (acciones) de alcanzar p son excluyentes con todas las formas de alcanzar q . Es importante decir que los mutex entre proposiciones son consecuencia directa de la propagación de los mutex entre acciones a lo largo del grafo de planificación. De esta forma cuando dos acciones dejan de ser mutex, el mutex entre sus efectos también desaparece (existe al menos una forma de generar los efectos que no es mutex). Obviamente, el grafo de planificación resultante proporciona más información que el correspondiente grafo relajado, al considerarse la interacción entre los efectos negativos de las acciones.

Ejemplo 13.6. En el problema de transporte definido en la Tabla 13.1, los mutex que aparecen en cada nivel son los que aparecen en la Tabla 13.2. Por ejemplo, en el nivel $A_{[1]}$ tenemos la pareja de mutex $\{mv(c1, cA, CB), cg(p1, c1, cA)\}$, al eliminar $mv(c1, cA, CB)$ el camión $c1$ de cA , algo que requiere la acción de carga. La propagación de los mutex del nivel $A_{[1]}$ hace que en $P_{[1]}$ la proposición $pos(c1, cB)$ sea mutex, entre otras, con $en(p1, c1)$, situación que desaparece en el nivel $P_{[2]}$, permitiendo la ejecución de $drg(p1, c1, cB)$ en $A_{[3]}$. Además, también se detectan mutex que, a priori, no son directamente observables como la pareja $\{mv(c1, cA, CB), drg(p1, c1, cB)\}$ en $A_{[3]}$ (evidentemente, en el problema real, estas dos acciones no se pueden ejecutar a la vez). Es justamente esta propagación de mutex la que permite generar un grafo de planificación más informado y preciso. En Graphplan, si las precondiciones de una acción son mutex, ésta no se genera. En la Figura 13.4, la acción $drg(p1, c1, cB)$ y el objetivo $pos(c1, cB)$ aparecen en el nivel 2, mientras que con la propagación de los mutex no aparecen hasta el nivel 3, estando más acorde con el problema real. Por tanto, el número de niveles del grafo de planificación de Graphplan nunca será menor que el de uno relajado.

Aunque las reglas que aplica Graphplan no garantizan encontrar todas las relaciones de exclusión, sí son lo suficientemente potentes para determinar cuándo dos acciones o dos proposiciones no pueden darse simultáneamente. De hecho, la condición de terminación para la construcción del grafo de planificación en Graphplan es la de alcanzar un nivel en el que los objetivos del problema dejan de ser mutex dos a dos².

13.6.2.2 Algoritmo de Graphplan

El esquema de funcionamiento de Graphplan se muestra en el Algoritmo 13.3. Tras construir el grafo de planificación, se realiza la etapa de extracción de un plan de igual

²Esta condición de terminación es necesaria para poder encontrar un plan, pero no suficiente. Por lo tanto, no existe garantía de que un plan válido se pueda encontrar desde el nivel actual (los mutex se calculan por parejas, pero puede existir un mutex que involucre a más de dos proposiciones/acciones que no se haya detectado).

longitud a la del grafo, que es donde se encuentra la verdadera complejidad temporal de la planificación. Si no se encuentra un plan en el grafo actual, puede ocurrir que no sea posible encontrar solución y, por tanto el problema no tenga solución, o sea necesario extender un nuevo nivel del grafo y repetir la etapa de búsqueda. Básicamente, *Extraer_Plan* realiza una búsqueda regresiva basada en backtracking cronológico extrayendo un plan como un *flujo de acciones* del grafo de planificación. Para ello selecciona un conjunto de acciones que resuelven los objetivos en un nivel t , luego trata de resolver las precondiciones de dichas acciones en el nivel $t - 1$, y así sucesivamente hasta llegar al nivel 0. Por otro lado, *No_Es_Possible_Encontrar_Plan* consiste en comprobar si se ha alcanzado un nivel que no aporta nada respecto al anterior, porque: 1) las proposiciones, acciones y mutex son idénticos, y 2) el mismo conjunto de objetivos sigue siendo irresoluble.

Nivel	Relaciones mutex
$A_{[1]}$	$\{\text{mv}(c1, cA, cB)\} \times \{\text{cg}(p1, c1, cA)\}$
$P_{[1]}$	$\{\text{pos}(c1, cB)\} \times \{\text{pos}(c1, cA), \text{en}(p1, c1)\}$ $\{\text{pos}(p1, cA)\} \times \{\text{en}(p1, c1)\}$
$A_{[2]}$	$\{\text{mv}(c1, cB, cA)\} \times \{\text{mv}(c1, cA, cB), \text{cg}(p1, c1, cA), \text{dcg}(p1, c1, cA)\}$ $\{\text{mv}(c1, cA, cB)\} \times \{\text{cg}(p1, c1, cA), \text{dcg}(p1, c1, cA)\}$ $\{\text{cg}(p1, c1, cA)\} \times \{\text{dcg}(p1, c1, cA)\}$
$P_{[2]}$	$\{\text{pos}(c1, cB)\} \times \{\text{pos}(c1, cA)\}$ $\{\text{pos}(p1, cA)\} \times \{\text{en}(p1, c1)\}$
$A_{[3]}$	$\{\text{mv}(c1, cB, cA)\} \times \{\text{mv}(c1, cA, cB), \text{cg}(p1, c1, cA), \text{dcg}(p1, c1, cB), \text{dcg}(p1, c1, cA)\}$ $\{\text{mv}(c1, cA, cB)\} \times \{\text{cg}(p1, c1, cA), \text{dcg}(p1, c1, cB), \text{dcg}(p1, c1, cA)\}$ $\{\text{cg}(p1, c1, cA)\} \times \{\text{dcg}(p1, c1, cB), \text{dcg}(p1, c1, cA)\}$ $\{\text{dcg}(p1, c1, cB)\} \times \{\text{dcg}(p1, c1, cA)\}$
$P_{[3]}$	$\{\text{pos}(c1, cB)\} \times \{\text{pos}(c1, cA)\}$ $\{\text{pos}(c1, cA)\} \times \{\text{pos}(p1, cB)\}$ $\{\text{pos}(p1, cA)\} \times \{\text{en}(p1, c1), \text{pos}(p1, cB)\}$ $\{\text{en}(p1, c1)\} \times \{\text{pos}(p1, cB)\}$

Tabla 13.2: Mutex resultantes (complementarios al grafo de planificación) para el ejemplo de transporte. Para simplificar, los mutex en los que se involucran acciones *no-op* no se muestran.

Con este esquema, y para el problema de ejemplo que nos ocupa, *Graphplan* encontraría el siguiente plan de longitud 3 $\{\{\text{cg}(p1, c1, cA)\}_{[1]}, \{\text{mv}(c1, cA, cB)\}_{[2]}, \{\text{dcg}(p1, c1, cB)\}_{[3]}\}$. *Graphplan* propone un algoritmo de planificación correcto, completo y óptimo. La correctitud y completitud son consecuencia de la construcción del grafo de planificación y de la completitud de la búsqueda basada en backtracking. La optimalidad se garantiza en términos de pasos de planificación (niveles), pero no en número de acciones ya que múltiples acciones se pueden ejecutar en paralelo en cada paso.

13.6.2.3 Graphplan vs. resolución de un CSP

El proceso de búsqueda llevado a cabo por *Graphplan* se puede considerar equivalente al de resolución de un CSP. El primer paso consiste en codificar el grafo

de planificación como un CSP dinámico, y el segundo en utilizar cualquiera de las técnicas estudiadas en el capítulo 10 para resolverlo.

Algoritmo 13.3 Esquema básico de Graphplan

```

1:  $GP \leftarrow \text{Construir\_Grafo\_Planificacion}(\mathcal{I})$  {Construcción del grafo de planificación}
2: bucle
3:    $\Pi \leftarrow \text{Extraer\_Plan}(GP, \text{Longitud}(GP), \mathcal{G})$  {Búsqueda}
4:   si  $\Pi$  es solución entonces
5:     Devuelve  $\Pi$  {Éxito}
6:   si no
7:     si  $No\_Es\_Posible\_Encontrar\_Plan(GP, \text{Longitud}(GP))$  entonces
8:       Devuelve fallo {Garantía de terminación si no existe plan}
9:     si no
10:     $GP \leftarrow \text{Extender\_Un\_Nivel}(GP)$  {Construcción del grafo de planificación}
11:  fin si
12: fin si
13: fin bucle

```

Para convertir un grafo de planificación en un CSP dinámico basta con aplicar las siguientes reglas sencillas. Las proposiciones (incluyendo objetivos) constituyen las variables, y las acciones que las generan (relevantes) constituyen los dominios de dichas variables. Las relaciones de exclusión entre acciones se representan mediante *restricciones de mutex*: si las proposiciones p y q son generadas por las acciones a y b , respectivamente, que son mutex, entonces se genera $p=a \Rightarrow q \neq b$. La asignación de valores a las variables es un proceso dinámico porque cada asignación en un determinado nivel activa otras variables en un nivel previo (*restricción de activación*): utilizar la acción a para satisfacer p en un nivel hace que las precondiciones de a se activen en el nivel previo, $\forall p \in Efe^+(a): p=a \Rightarrow \text{Activa}(\text{Pre}(a))$. Inicialmente, sólo las variables correspondientes a los objetivos del problema están activas. En la Tabla 13.3 se muestra el CSP dinámico correspondiente a los niveles $P_{[0]}$, $P_{[1]}$ y $A_{[1]}$ del grafo de planificación de la Figura 13.4, teniendo en cuenta los mutex de la Tabla 13.2. Esta simple conversión en un CSP facilita su resolución mediante técnicas y criterios heurísticos eficientes ampliamente estudiados.

Variables	$pos(c1, cB) = \{\text{mv}(c1, cA, cB)\}, pos(c1, cA) = \{\text{no-op}(pos(c1, cA))\},$ $pos(p1, cA) = \{\text{no-op}(pos(p1, cA))\}, en(p1, c1) = \{cg(p1, c1, cA)\}$
R. Mutex	$pos(c1, cB) = \text{mv}(c1, cA, cB) \Rightarrow en(p1, c1) \neq cg(p1, c1, cA) \wedge$ $pos(c1, cA) \neq \text{no-op}(pos(c1, cA))$ $pos(c1, cA) = \text{no-op}(pos(c1, cA)) \Rightarrow pos(c1, cB) \neq \text{mv}(c1, cA, cB)$ $pos(p1, cA) = \text{no-op}(pos(p1, cA)) \Rightarrow en(p1, c1) \neq cg(p1, c1, cA)$ $en(p1, c1) = cg(p1, c1, cA) \Rightarrow pos(c1, cB) \neq \text{mv}(c1, cA, cB) \wedge$ $pos(p1, cA) \neq \text{no-op}(pos(p1, cA))$
R. Activación	$pos(c1, cB) = \text{mv}(c1, cA, cB) \Rightarrow \text{Activa}(pos(c1, cA))$ $pos(c1, cA) = \text{no-op}(pos(c1, cA)) \Rightarrow \text{Activa}(pos(c1, cA))$ $pos(p1, cA) = \text{no-op}(pos(p1, cA)) \Rightarrow \text{Activa}(pos(p1, cA))$ $en(p1, c1) = cg(p1, c1, cA) \Rightarrow \text{Activa}(pos(c1, cA) \wedge pos(p1, cA))$

Tabla 13.3: Ejemplo de CSP dinámico resultante para un fragmento de un grafo de planificación de Graphplan.

13.6.3 Heurísticas basadas en grafos de planificación

Los grafos de planificación proporcionan una estructura eficiente que permite calcular estimaciones heurísticas útiles para guiar la búsqueda en planificadores progresivos basados en estados. La utilización de estos grafos conlleva la relajación de los efectos negativos de las acciones para que el conjunto de proposiciones alcanzables sea siempre creciente³. La idea consiste en construir desde cada estado un grafo de planificación hasta el objetivo. De esta forma, el coste $h(p)$ de alcanzar una proposición p desde un estado inicial se puede estimar como el primer nivel en el que p aparece en el grafo de planificación. Por ejemplo según la Figura 13.4, $h(\text{pos}(c1, cA)) = 0$ y $h(\text{pos}(p1, cB)) = 2$. Aunque esta heurística es admisible, también es poco informada pues los problemas de planificación implican generalmente varios objetivos. No obstante, extender esta estimación para aplicarla a un conjunto de objetivos, como por ejemplo $\mathcal{G} = \{\text{pos}(c1, cA), \text{pos}(p1, cB)\}$, resulta bastante intuitivo, apareciendo la siguiente familia de heurísticas:

- h_{sum} , que estima el coste como la suma de los costes individuales de cada una de las proposiciones: $h_{\text{sum}}(\mathcal{G}) = \sum_{g \in \mathcal{G}} h(g) = 0 + 2 = 2$. h_{sum} es no admisible porque no tiene en cuenta que una acción genere varios objetivos simultáneamente, pero funciona muy bien cuando los objetivos son independientes entre sí y el problema se puede descomponer fácilmente.
- $h_{\text{máx}}$, que estima el coste como el máximo de los costes individuales: $h_{\text{máx}}(\mathcal{G}) = \max_{g \in \mathcal{G}} h(g) = \max(0, 2) = 2$. $h_{\text{máx}}$ es admisible pero cuando existen muchos objetivos no siempre es suficientemente informada al descartarse el resto de los costes individuales.
- $h_{\text{máx}2}$, que estima el coste como el máximo nivel en el que coexisten cada par de objetivos: $h_{\text{máx}2}(\mathcal{G}) = \max_{\{g_1, g_2\} \in \mathcal{G}} h(g_1 \wedge g_2) = \max(2) = 2$. $h_{\text{máx}2}$ sigue siendo admisible y, en general, es mucho más informada que $h_{\text{máx}}$ al tener en cuenta el coste máximo de alcanzar cada par de objetivos.
- $h_{\text{máx}k}$, que generaliza $h_{\text{máx}2}$, y estima el coste como el máximo nivel en el que coexiste cada tupla de k -objetivos: $h_{\text{máx}k}(\mathcal{G}) = \max_{\{g_1, g_2 \dots g_k\} \in \mathcal{G}} h(g_1 \wedge g_2 \wedge \dots \wedge g_k)$. A pesar de que $h_{\text{máx}k}$ es admisible y muy informada, es muy costosa de calcular y resulta poco práctica, por lo que la mayoría de las estimaciones utiliza valores $k \leq 2$.

El razonamiento sobre mutex que lleva a cabo Graphplan en el grafo de planificación, aunque ligeramente más costoso, también resulta beneficioso para la estimación de las heurísticas anteriores. Si se considera la información sobre mutex de la Tabla 13.2, el nuevo valor para las heurísticas anteriores es: $h_{\text{sum}}(\mathcal{G}) = 0 + 3 = 3$, $h_{\text{máx}}(\mathcal{G}) = \max(0, 3) = 3$ y $h_{\text{máx}2}(\mathcal{G}) = \max(4) = 4$. Puede observarse cómo el cálculo de los mutex proporciona información más precisa en los tres casos al tener

³Realizar una estimación heurística teniendo en cuenta los efectos negativos (las proposiciones ya generadas pueden eliminarse) sería tan costoso como resolver el problema original, y la estimación dejaría de tener utilidad práctica.

en cuenta sólo las combinaciones factibles que permiten alcanzar los objetivos. Por ejemplo, la estimación de $h_{\max 2}$ coincide con el valor real; según la Tabla 13.2 dicho par de objetivos es mutex en el nivel $P_{[3]}$, y no deja de serlo hasta $P_{[4]}$ (tal y como ocurre en el plan solución).

Esta familia de heurísticas es de gran ayuda en planificadores progresivos, aunque son costosas al tener que reconstruir un grafo de planificación desde cada estado hasta el objetivo del problema. No obstante, es fácil adaptar dichas heurísticas para su utilización en planificadores regresivos simplemente construyendo el grafo de planificación desde el estado inicial una **única** vez, estimando el coste a cada una de las proposiciones del problema, y utilizando dichas estimaciones tantas veces como sea necesaria. Así, cada vez que la búsqueda regresiva deba decidirse por uno u otro estado hará uso de este valor y elegirá el que presente un menor coste hasta la situación inicial.

La calidad de las heurísticas anteriores se puede mejorar todavía más, calculando la estimación no sobre un grafo de planificación sino sobre un plan. De nuevo, para calcular el plan se mantiene la misma relajación de eliminación de efectos negativos, por lo que el plan es realmente un **plan relajado** sin interacciones negativas que se puede calcular en tiempo polinómico (véase el ejercicio propuesto 5). La información que proporciona este plan es mucho más precisa y variada. En lugar de estimar el coste como número de pasos de planificación, al disponerse de un plan se puede calcular el número de acciones del mismo, o se puede asignar un coste (temporal, económico, etc.) a cada acción y calcular el coste real del plan. Estas estimaciones, aunque bien informadas, suelen ser no admisibles, pues encontrar un plan relajado óptimo resulta tan costoso como encontrar el propio plan. Aun así, los resultados que se obtienen ayudan notablemente a los planificadores y les permiten abordar problemas de elevada complejidad.

13.7 Planificación basada en satisfacibilidad

La idea de abordar un problema de planificación como un problema basado en satisfacibilidad surge con el objetivo de reutilizar técnicas basadas en satisfacción de fórmulas proposicionales para su aplicación a planificación. El funcionamiento básico se muestra en el Algoritmo 13.4. Un sistema de traducción toma como entrada un problema de planificación con una longitud estimada y genera un conjunto de fórmulas lógicas o axiomas. A continuación, una serie de algoritmos muy eficientes busca una asignación factible (*modelo*) que satisface las fórmulas anteriores y, de ser así, ésta se traduce en un plan solución.

Algoritmo 13.4 Esquema básico de planificación basada en satisfacibilidad

- 1: $\text{formulas} \leftarrow \text{Traducir_a_SAT}(\mathcal{T} + \text{Acciones} + \mathcal{G}, \text{longitud})$
 - 2: $\text{modelo} \leftarrow \text{Algoritmo_Resolucion_SAT}(\text{formulas})$
 - 3: **si** \exists *modelo* **entonces**
 - 4: Devuelve *Extraer_Plan*(*modelo*) {Éxito, se ha encontrado un plan con la longitud dada}
 - 5: **si no**
 - 6: Devuelve fallo {No existe solución con la longitud dada}
 - 7: **fin si**
-

La traducción del problema de planificación en fórmulas lógicas es un paso clave, pues la complejidad de los algoritmos de resolución depende en gran medida del número de fórmulas y su longitud. Básicamente, esta traducción debe codificar los estados y las transiciones entre ellos producidas por las acciones. En la Tabla 13.4 se muestra dicha codificación para nuestro problema ejemplo. Un estado se codifica como una fórmula lógica con conjunciones y/o disyunciones de predicados, y debe estar completamente especificado, indicando los predicados que son ciertos y los que no (véase el estado \mathcal{I}), excepto cuando hay predicados cuyo valor es desconocido o irrelevante (en \mathcal{G} la posición del camión $c1$ es indiferente). Además, los estados evolucionan en el tiempo, por lo que es necesario extender cada predicado para indicar que se cumple en un instante de tiempo pero no en otro (por ejemplo el predicado $\text{pos}(c1, cB)_{[0]}$ debe ser falso en el instante 0 en \mathcal{I} , pero $\text{pos}(p1, cB)_{[2]}$ debe ser cierto en el instante 2 en \mathcal{G}). La principal dificultad que aparece aquí es que también los objetivos deben estar asociados con un instante de tiempo que viene marcado por el parámetro *longitud* del plan (véase el Algoritmo 13.4). En un esquema de resolución general, se comienza desde una *longitud* mínima y se va incrementando iterativamente hasta encontrar un plan o hasta alcanzar una *longitud* máxima⁴. En el ejemplo de la Tabla 13.4, los objetivos se han asociado con el instante 2, tal y como marca el grafo de planificación de la Figura 13.4. Las acciones se codifican como fórmulas que causan transiciones y deben asociarse con todos los instantes de tiempo en los que cada acción puede ejecutarse⁵; es decir, si existen dos instantes de tiempo posibles existirá una fórmula para cada instante. No obstante, la codificación de las acciones es un poco más elaborada pues requiere tres tipos de fórmulas por acción (que aparecen etiquetadas como F1, F2 y F3 en la Tabla 13.4): F1) las que codifican la propia transición, indicando la acción, precondiciones y efectos ($a_i \Rightarrow \text{Pre}(a_{i-1}) \wedge \text{Efe}(a_i)$), F2) las que garantizan que una acción sólo cambia lo que está en sus efectos (*problema marco*) o, dicho de otra forma, que si un efecto cambia lo hace por la ejecución de la acción que lo tiene como efecto ($(\neg p_i \wedge p_{i+1} \Rightarrow (\bigvee_{a|p \in \text{Efe}^+(a)} a_{i+1})) \wedge (p_i \wedge \neg p_{i+1} \Rightarrow (\bigvee_{a|p \in \text{Efe}^-(a)} a_{i+1}))$), y F3) las que impiden que dos acciones mutuamente excluyentes se ejecuten simultáneamente ($\neg a_i \vee \neg b_i$).

Una vez finalizada la conversión del problema en fórmulas lógicas se ejecutaría el algoritmo de resolución basado en satisfacibilidad. Con el valor actual de *longitud*=2 no se encontraría una solución, siendo necesario extenderlo hasta el instante 3, donde se encontraría la asignación factible $\{\text{cg}(p1, c1, cA)_{[1]}, \text{mv}(c1, cA, cB)_{[2]}, \text{dcg}(p1, c1, cB)_{[3]}\}$ de la que se puede extraer un plan solución.

13.8 Planificación para el mundo real

La aplicación de las técnicas de planificación a problemas del mundo real está aumentando a un ritmo considerable. Actualmente, existen decenas de escenarios reales en los que la planificación resulta una herramienta muy útil:

⁴ Algunos enfoques utilizan el último nivel del grafo de planificación correspondiente para iniciar el valor de la *longitud* mínima, evitando iteraciones innecesarias.

⁵ De nuevo, el grafo de planificación correspondiente proporciona información acerca de todos los posibles instantes de tiempo en los que las acciones pueden ejecutarse.

Estado \mathcal{I}	$\text{pos}(c1, cA)_{[0]} \wedge \text{pos}(p1, cA)_{[0]} \wedge \neg\text{pos}(c1, cB)_{[0]} \wedge \neg\text{en}(p1, c1)_{[0]} \wedge \neg\text{pos}(p1, cB)_{[0]}$
Estado \mathcal{G}	$\text{pos}(p1, cB)_{[2]} \wedge \neg\text{pos}(p1, cA)_{[2]} \wedge \neg\text{en}(p1, c1)_{[2]}$
Acciones	<p>F1. $\text{mv}(c1, cA, cB)_{[1]} \Rightarrow \text{pos}(c1, cA)_{[0]} \wedge \neg\text{pos}(c1, cB)_{[0]} \wedge \text{pos}(c1, cB)_{[1]} \wedge \neg\text{pos}(c1, cA)_{[1]}$ F2. $\neg\text{pos}(c1, cB)_{[0]} \wedge \text{pos}(c1, cB)_{[1]} \Rightarrow \text{mv}(c1, cA, cB)_{[1]}$ F2. $\text{pos}(c1, cA)_{[0]} \wedge \neg\text{pos}(c1, cA)_{[1]} \Rightarrow \text{mv}(c1, cA, cB)_{[1]}$ F3. $\neg\text{mv}(c1, cA, cB)_{[1]} \vee \neg\text{cg}(p1, c1, cA)_{[1]}$ F1. $\text{cg}(p1, c1, cA)_{[1]} \Rightarrow \text{pos}(c1, cA)_{[0]} \wedge \text{pos}(p1, cA)_{[0]} \wedge \text{en}(p1, c1)_{[1]} \wedge \neg\text{pos}(p1, cA)_{[1]}$ F2. $\text{pos}(p1, cA)_{[0]} \wedge \neg\text{pos}(p1, cA)_{[1]} \Rightarrow \text{cg}(p1, c1, cA)_{[1]}$ F2. $\neg\text{en}(p1, c1)_{[0]} \wedge \text{en}(p1, c1)_{[1]} \Rightarrow \text{cg}(p1, c1, cA)_{[1]}$ F3. $\neg\text{cg}(p1, c1, cA)_{[1]} \vee \neg\text{mv}(c1, cA, cB)_{[1]}$ F1. $\text{dcg}(p1, c1, cB)_{[2]} \Rightarrow \text{pos}(c1, cB)_{[1]} \wedge \text{en}(p1, c1)_{[1]} \wedge \neg\text{en}(p1, c1)_{[2]} \wedge \text{pos}(p1, cB)_{[2]}$ F2. $\text{en}(p1, c1)_{[1]} \wedge \neg\text{en}(p1, c1)_{[2]} \Rightarrow \text{dcg}(p1, c1, cB)_{[2]} \vee \text{dcg}(p1, c1, cA)_{[2]}$ F2. $\neg\text{pos}(p1, cB)_{[1]} \wedge \text{pos}(p1, cB)_{[2]} \Rightarrow \text{dcg}(p1, c1, cB)_{[2]}$ F3. $(\neg\text{dcg}(p1, c1, cB)_{[2]} \vee \neg\text{mv}(c1, cB, cA)_{[2]}) \wedge (\neg\text{dcg}(p1, c1, cB)_{[2]} \vee \neg\text{mv}(c1, cA, cB)_{[2]}) \wedge (\neg\text{dcg}(p1, c1, cB)_{[2]} \vee \neg\text{cg}(p1, c1, cA)_{[2]}) \wedge (\neg\text{dcg}(p1, c1, cB)_{[2]} \vee \neg\text{dcg}(p1, c1, cA)_{[2]})$</p>

Tabla 13.4: Ejemplo de codificación de un problema de planificación con las acciones $\text{mv}(c1, cA, cB)$, $\text{cg}(p1, c1, cA)$ y $\text{dcg}(p1, c1, cB)$ mediante fórmulas lógicas. Para simplificar sólo se han representado las fórmulas asociadas a un instante de tiempo.

- **Control y navegación de robots autónomos.** Este campo es uno de los que motivó la investigación inicial en planificación de trayectorias y exploración del entorno para identificar objetos. En la actualidad, la planificación también se está usando con éxito en entornos portuarios, fábricas de automóviles e incluso en misiones aeroespaciales de la NASA y la ESA (*Agencia Espacial Europea*) donde la mayoría de las tareas a realizar se planifican automáticamente sin intervención humana [Jónsson y otros, 2000].
- **Planificación de rutas.** La generación de rutas logísticas para empresas de transporte o incluso la elaboración de los planes de rutas de los servicios de transporte público utilizan técnicas de planificación para realizar la asignación de vehículos y personal de forma automatizada [Ghallab y otros, 2004].
- **Entornos simulados.** Las técnicas de planificación inteligentes se han aplicado en el diseño de agentes para programas de entrenamiento e incluso en juegos comerciales de acción, cartas, etc. [Nareyek, 2004].
- **Entornos de red.** Con el auge de las redes la planificación se ha comenzado a utilizar en muchos de los buscadores para realizar consultas y en la construcción de servicios Web distribuidos [Knoblock, 2003]. Otra aplicación importante es la organización de flujos de tareas (*workflows*) en redes computacionales [Gil y otros, 2004].
- **Situaciones catastróficas y manejo de crisis.** Desde hace unos años las técnicas de planificación están dando soporte a los planes de actuación ante situaciones desastrosas, como por ejemplo vertidos de petróleo, incendios forestales, evacuaciones urbanas y rescates, etc. [Asunción y otros, 2005; Gervasio y otros, 1998; RoboCup Rescue Technical Committee, 2000].

- **Aplicaciones militares.** Las técnicas de planificación han tomado parte en el diseño de importantes campañas militares y en la toma de decisiones logísticas y estratégicas de posicionamiento de tropas [Chun, 1999].

Tal y como puede observarse, son muchos los escenarios que se benefician de las técnicas de planificación pero, para ello, es necesario ir más allá de la planificación clásica. Los problemas del mundo real requieren características que exceden del modelo clásico y de sus asunciones (véase la sección 13.1), como gestión de duraciones y recursos, optimización multicriterio, abstracción de acciones o trabajo con incertidumbre, entre otras. El resto de esta sección proporciona una breve introducción a nuevas técnicas de planificación que solventan las principales limitaciones de la planificación clásica.

13.8.1 Planificación numérica: tiempo + recursos

13.8.1.1 Planificación temporal

En los problemas reales de planificación no es aceptable que todas las acciones tengan la misma duración. Obviamente, la duración de una acción $mv(c1, cA, cB)$ no tiene por qué ser igual a la de $mv(c2, cA, cC)$ pues la velocidad del camión o distancia entre las ciudades puede ser distinta. Por lo tanto, la primera asunción de la planificación clásica que relajaremos es la A5: las acciones ahora tienen distinta duración y es el planificador el que debe gestionar las características temporales de forma explícita (**planificación temporal**). La resolución de un problema de planificación temporal incrementa la complejidad del problema porque no sólo estriba en la correcta elección de la secuencia de acciones que alcanza el objetivo, sino también en la correcta elección del instante de ejecución apropiado para las mismas. Adicionalmente, esta doble elección debe realizarse con el objetivo de generar planes de mínima duración. Es decir, en un contexto de planificación temporal el criterio de optimización de los planes se centra en minimizar la duración del plan, en lugar del número de pasos o acciones del mismo.

En la planificación temporal las acciones dejan de ser instantáneas y su duración se puede extender a lo largo del tiempo. Esto hace que el modelo simple (conocido como modelo *conservativo*) de acción en el que las precondiciones deben mantenerse durante toda la ejecución de la acción y los efectos se generan sólo al final de la misma resulte un poco limitado. Para amoldarse a la duración de la acción (Dur) y hacer frente a esta limitación, el modelo de acción se extiende teniendo en cuenta condiciones locales: precondiciones (Pre), postcondiciones (Post) y condiciones *invariantes* (Inv) que deben satisfacerse antes, después y durante la ejecución de la acción, respectivamente. Adicionalmente, los efectos de las acción también se dividen en efectos *iniciales* (EfeI) y *finales* (EfeF) que se generan al principio y final de la acción, respectivamente⁶. Esto permite disponer de acciones más expresivas como las que se muestran en la Tabla 13.5, y modelar con mayor precisión los cambios que

⁶Éste es el modelo de acción propuesto en las últimas versiones de PDDL. Además de este modelo existen otros menos comunes, pero más expresivos, que permiten que tanto las condiciones como los efectos se definan como ventanas temporales a lo largo de la ejecución de la acción.

las mismas provocan en el mundo, dando lugar a planes con más oportunidades de concurrencia entre acciones. Por ejemplo, en la Figura 13.5 el operador `mv` tiene como *postcondición* que exista un almacén en la ciudad de destino y como condición invariante que el camión tenga combustible. Para obtener este combustible se añade un nuevo operador de repostar (`rep`) que requiere la presencia de un operario sólo al inicio (precondición) y final (postcondición) del mismo y, lógicamente, que el camión permanezca en esa posición de forma invariante. Análogamente, los operadores `cg` y `dgc` también modelan condiciones y efectos locales, acercándose más al problema real.

<code>mv(?cam,?ori,?des)</code> Dur: <code>dur-mv(?cam,?ori,?des)</code> Pre: <code>pos(?cam,?ori)</code> Post: <code>almacen(?des)</code> Inv: <code>combustible(?cam)</code> EfeI: <code>¬pos(?cam,?ori)</code> EfeF: <code>pos(?cam,?des)</code> <code>¬combustible(?cam)</code>	<code>rep(?cam,?ciu)</code> Dur: <code>dur-rep(?cam,?ciu)</code> Pre: <code>operario(?ciu)</code> Post: <code>operario(?ciu)</code> Inv: <code>pos(?cam,?ciu)</code> EfeI: <code>∅</code> EfeF: <code>combustible(?cam)</code>
<code>cg(?paq,?cam,?ciu)</code> Dur: <code>dur-cg(?paq,?cam,?ciu)</code> Pre: <code>pos(?paq,?ciu)</code> Post: <code>∅</code> Inv: <code>pos(?cam,?ciu)</code> EfeI: <code>¬pos(?paq,?ciu)</code> EfeF: <code>en(?paq,?cam)</code>	<code>dgc(?paq,?cam,?ciu)</code> Dur: <code>dur-dgc(?paq,?cam,?ciu)</code> Pre: <code>en(?paq,?cam)</code> Post: <code>∅</code> Inv: <code>pos(?cam,?ciu)</code> EfeI: <code>¬en(?paq,?cam)</code> EfeF: <code>pos(?paq,?ciu)</code>

Tabla 13.5: Ejemplo de operadores utilizando un modelo de acciones extendido para manejar duraciones y condiciones/efectos locales.

Los planificadores temporales gestionan la información temporal mediante una representación explícita del tiempo y algoritmos de búsqueda para asignar las acciones en el tiempo. Inicialmente, la planificación temporal se abordó bajo una aproximación de orden parcial, basándose principalmente en las técnicas de menor compromiso; la idea es retrasar la instanciación temporal de las acciones tanto como sea posible para reducir la complejidad del problema. En otras aproximaciones, el manejo de tiempo se consigue gracias a la inclusión de una red temporal que registra las restricciones temporales entre acciones y permite la definición de restricciones sobre la duración de las acciones y del plan. Gracias a esta representación temporal, el planificador puede manejar relaciones cualitativas (restricciones de orden y enlaces causales) y cuantitativas (instantes de comienzo/fín y duraciones). La aproximación basada en grafos también se ha aplicado a la planificación temporal, extendiendo el razonamiento sobre mutex, que es ahora más elaborado, y generalizando el grafo de planificación para contemplar el modelo más expresivo de las acciones con duración. En este caso, los niveles del grafo no representan pasos de planificación, sino verdaderos instantes de tiempo, lo que incrementa el número de niveles del grafo y hace la búsqueda más costosa (el espacio de búsqueda es ahora mayor). Por último, la aproximación que posiblemente ha permitido un mayor avance en el campo de la planificación temporal ha sido la de planificación heurística, pero ésta ha abordado conjuntamente el problema de la planificación con recursos.

13.8.1.2 Planificación con recursos

Habitualmente, las acciones de un plan requieren un conjunto de recursos para llevar a cabo sus tareas. Se puede decir que un recurso es cualquier elemento (generalmente limitado y que hay que compartir a lo largo del tiempo) necesario para poder ejecutar las acciones. Un recurso puede representar a una entidad material (camión, máquina, combustible, etc.) o personal (conductor, operario, tripulación, etc.). Aunque tradicionalmente la gestión de recursos se ha relegado al proceso de **scheduling**, en planificación el uso no simultáneo de recursos se resuelve mediante métodos *artificiales* que consisten, básicamente, en introducir nuevas proposiciones para representar la disponibilidad y uso exclusivo de recursos, como por ejemplo **camion-libre**, **conductor-disponible**, etc. Sin embargo, existen recursos que requieren una representación basada en valores numéricos continuos (por ejemplo, combustible, energía, tiempo, etc.) que no se pueden representar bajo el modelo proposicional impuesto por la planificación clásica. Por lo tanto, para abordar problemas reales con este tipo de recursos es necesario relajar la asunción A4 (enfoque proposicional): las acciones ahora pueden trabajar con variables numéricas cuyo dominio es continuo. Esto permite modelar restricciones más complejas sobre las variables y acciones: 1) precondiciones como **(nivel-combustible(c1) ≥ 100.5)** o **(espacio-ocupado(c1) < 50)**, y 2) efectos como **(decrementar nivel-combustible(c1) 12.25)** o **(incrementar beneficio 1.5)**. A este tipo de planificación se le conoce como **planificación numérica** y conlleva dos claras ventajas:

- Esta planificación subsume a la planificación temporal, pues basta considerar al tiempo como una variable numérica más. Así, al igual que una acción puede incrementar el total de combustible consumido, también puede incrementar el tiempo (duración) total del plan en función de su propia duración.
- Esta planificación permite llevar a cabo una **optimización multicriterio**. Ahora es posible expresar una función objetivo del estilo **MINIMIZAR ((2 * duracion-plan) + (0.005 * combustible-consumido(c1)))** en el que se tienen en cuenta varios factores ponderados para evaluar y minimizar el coste del plan, obteniendo planes de mejor calidad.

La planificación numérica (incluyendo tiempo y recursos) resulta bastante compleja por lo que es necesario abordarla bajo una aproximación heurística. En muchos casos las estimaciones heurísticas se calculan a partir de grafos de planificación y planes relajados. Estas estimaciones suelen representar cotas optimistas y pesimistas que se combinan con técnicas de **búsqueda local** y **descomposición de problemas** para realizar una exploración inteligente del espacio de búsqueda del problema.

A pesar del intento de incluir características propias de scheduling, como la gestión del tiempo y recursos, dentro del proceso de planificación todavía existen algunas limitaciones. La tipología de restricciones temporales y recursos de un problema real es muy variada e incluir por completo su razonamiento en la planificación resulta excesivamente complejo. Por ejemplo, supongamos una planta de robots industriales. En función del robot a utilizar (recurso) y de las acciones a realizar (ensamblar, pintar,

soldar, etc.) se deberán satisfacer unas restricciones u otras, con distintos tiempos de preparación del recurso y de utilización (la preparación del robot requiere distintos tiempos y acciones para ensamblar piezas que para pintarlas, o los tiempos de espera entre dos acciones de soldar serán distintos que entre dos acciones de pintar). Por tanto, uno de los campos de trabajo más activos dentro de la planificación es su **integración** con scheduling. La idea consiste en dirigir los esfuerzos de planificación en la elaboración de los planes, mientras que los esfuerzos de scheduling se enfocan en la validación y satisfacción de las restricciones complejas del problema. El objetivo es aunar esfuerzos para resolver problemas reales complejos de una forma integrada, compartiendo información y criterios heurísticos de optimización.

13.8.2 Planificación jerárquica

En muchos problemas reales, la *granularidad* que se espera de los planes no tiene que ser muy alta, al menos en un primer momento. Es decir, a menudo resulta más interesante generar un plan inicial compuesto por tareas de alto nivel para luego ir desglosándolo en acciones más simples. Esto es especialmente útil en problemas muy complejos, donde planificar desde cero todas las acciones de bajo nivel puede resultar una tarea muy costosa. Por ejemplo, en un escenario de transporte con decenas de paquetes es preferible contar, en un primer nivel, con una planificación más abstracta sobre las rutas para transportar los paquetes. Posteriormente, en un segundo nivel se podrá descomponer cada una de estas rutas en las acciones correspondientes de carga, movimiento, repostaje y descarga sobre camiones determinados. De ser necesario, este proceso de descomposición puede continuar hasta llegar a las acciones de más bajo nivel. La idea subyacente es la de establecer una jerarquía de acciones que permita descomponer un plan desde las acciones más genéricas hasta las más primitivas. Para ser capaces de manejar esta jerarquía es necesario relajar la asunción A6 de la planificación clásica que obliga a trabajar con acciones primitivas, es decir que no se pueden descomponer. Al modelo de planificación resultante se le conoce como **planificación jerárquica** o HTN (*Hierarchical Task Network*).

En la planificación jerárquica existen dos tipos de acciones: unas de más alto nivel denominadas **métodos** y unas de bajo nivel denominadas **acciones primitivas**. Al contrario que las acciones primitivas, los métodos se descomponen en subtareas que pueden ser otros métodos o acciones primitivas. Esta descomposición se define mediante una red de tareas, que establece la ordenación total o parcial de las subtareas de los métodos. Por ejemplo, en el problema de transporte podemos definir una tarea para transportar un paquete de una ciudad a otra (véase la Tabla 13.6). Esta tarea requerirá la ejecución de un método que se descompondrá, siguiendo una ordenación total, en las subtareas $\langle cg, mv, dcg \rangle$ que se corresponden con acciones primitivas.

El funcionamiento de un planificador HTN es muy similar al de un planificador clásico. El objetivo del problema consiste en aplicar una serie de tareas mediante la descomposición y planificación de una secuencia de métodos que, finalmente, se desglosará en las acciones primitivas. Por lo tanto, se trata de un procedimiento recursivo que va descomponiendo tareas no primitivas hasta llegar a un plan formado sólo por acciones primitivas, que son las que se pueden ejecutar directamente en el entorno.

Este tipo de planificación presenta dos ventajas: 1) el dominio del problema se describe en términos de acciones estructuradas jerárquicamente, resultando más intuitivo para el experto que modela el problema, y 2) la función del planificador consiste en refinar estas estructuras, generando las expansiones necesarias y simplificando la resolución del problema original.

Objetivos del problema	
Tarea	<code>transportar-paquete(?paq,?ori,?des)</code>
Descomposición jerárquica	
Método	<code>transportar(?paq,?cam,?ori,?des)</code>
Pre	<code>pos(?paq,?ori), pos(?cam,?ori)</code>
Subtareas	<code>(cg(?paq,?cam,?ori), mv(?cam,?ori,?des), dcg(?paq,?cam,?des))</code>

Tabla 13.6: Método `transportar(?paq,?cam,?ori,?des)`. Ejemplo de descomposición del método que se aplica para la tarea de transportar un paquete.

13.8.3 Planificación con incertidumbre

La planificación clásica asume que la información sobre el mundo es estática y totalmente observable (el planificador tiene un conocimiento total sobre el estado del sistema –asunciones A1 y A2), y que los efectos de las acciones son deterministas (la aplicación de una acción genera siempre los mismos efectos conocidos –asunción A3). Sin embargo, en los problemas reales esto no siempre es cierto, existiendo cierta incertidumbre. Esta incertidumbre puede aparecer en el entorno (mundo) del problema, en la percepción que el planificador tiene de ese entorno y/o en las propias acciones. En primer lugar, el modelo de entorno puede ser dinámico, existiendo agentes externos que lo modifican sin conocimiento del planificador. De hecho, esto también incluye los objetivos del problema que pueden cambiar o incluso volverse imposibles de satisfacer. En segundo lugar, la percepción que el planificador posee del entorno puede ser incompleta, desconociendo cierta información. Por último, la ejecución de las acciones no siempre es determinista y predecible: en ocasiones, la duración de las acciones fluctúa y no siempre tienen el éxito esperado, o incluso sólo existe una cierta probabilidad de éxito. Supongamos, por ejemplo, un escenario de planificación para la lucha contra incendios forestales. Es evidente que este problema tiene un elevado componente dinámico, pues el fuego (agente externo) altera el entorno (aparición y desaparición de nuevos focos que el planificador desconoce) y modifica constantemente los objetivos del problema. Además, las duraciones y efectos de las acciones de extinción son difícilmente predecibles y su probabilidad de éxito depende de la virulencia del fuego. Para gestionar esta incertidumbre se han planteado dos aproximaciones distintas:

- Construir planes suficientemente genéricos que contemplen dicha incertidumbre mediante la inclusión de acciones específicas de *sensorización* para adquirir la información incompleta, o mediante *monitorización* para observar el entorno y actuar en consecuencia. Más concretamente existen las siguientes técnicas:

1) **planificación conforme** para entornos parcialmente observables y sin posibilidad de monitorización, que generan planes para todos los posibles mundos a partir de distintos estados iniciales, 2) **planificación contingente** para generar planes condicionales o probabilísticos preparados para la aparición de *ciertos* imprevistos conocidos, y 3) **planificación reactiva** que, tras monitorizar el estado, recomienda las próximas acciones a ejecutar.

- Modificar y adaptar el plan a medida que se ejecuta. Esta técnica de **planificación y ejecución** alterna las etapas de planificación, monitorización durante la ejecución y replanificación en caso de que el entorno no se encuentre en el estado esperado. A este tipo de planificación también se le conoce como **planificación on-line**, y relaja la asunción A7 del modelo clásico.

13.9 Lecturas recomendadas

Desde sus orígenes, la planificación ha consistido en un proceso formal de resolución de problemas mediante búsqueda y técnicas de demostración de problemas, entre los que cabe destacar a GPS [Newell y Simon, 1963] y QA3 [Green, 1969], respectivamente. STRIPS [Fikes y Nilsson, 1971] supuso un importante punto de partida, tanto por ser el primer sistema práctico aplicado a la planificación de trayectorias de un robot, como por fijar las bases de un modelo de acciones que, con algunas extensiones como las que propone ADL [Pednault, 1989], todavía se sigue utilizando. Aunque la idea de STRIPS era la de mantener el espacio de búsqueda próximo al espacio de situaciones, pronto dio paso a nuevos planificadores más complejos, expresivos y con un mejor control sobre la búsqueda, como NOAH [Sacerdoti, 1977] y Nonlin [Tate, 1977]. En estos dos planificadores el orden de acciones no tiene que ser estrictamente total y lineal, sino que a menudo se mantienen en un orden parcial [Barret y Weld, 1994].

En paralelo a los avances teóricos, la planificación práctica también avanzaba con paso firme, apareciendo sistemas que ayudaban en la construcción de edificios o en las operaciones de vehículos espaciales [Yang, 1997]. Por ejemplo, Deviser [Vere, 1983], Sipe [Wilkins, 1988], ZENO [Penberthy, 1993; Penberthy y Weld, 1994] y parcPLAN [El-Kholy y Richards, 1996] incluyeron criterios de temporalidad en las acciones y planes. Sin embargo, estos sistemas resultan mucho más complejos y no tuvieron el mismo impacto que los basados en planificación clásica como SNLP [McAllester y Rosenblitt, 1991], que integra planificación de orden parcial [Barret y Weld, 1994] y una dependencia entre acciones mediante enlaces causales [McAllester y Rosenblitt, 1991; Weld, 1994]. Las propiedades formales de los POP fueron demostradas por UCPOP [Barret y otros, 1996; Penberthy y Weld, 1992], uno de los planificadores de orden parcial más conocidos que, junto con Prodigy [Fink y Veloso, 1994], aplicaba además técnicas de menor compromiso [Weld, 1994]. A medida que la complejidad de los problemas crecía, se hacía más necesario el desarrollo de técnicas de planificación abstracta y jerárquica [Erol y otros, 1994], como en ABSTRIPS [Sacerdoti, 1974], ABTWEAK [Yang y Tenenberg, 1990], ALPINE [Knoblock, 1994] y SIADEX [Asunción y otros, 2005; Castillo y otros, 2005]. Sin embargo, la complejidad no se origina sólo por el tamaño de los problemas, sino también por la expresividad requerida. En la

línea de unificar elementos de planificación y scheduling bajo un mismo enfoque cabe destacar a O-Plan [Currie y Tate, 1991] (utilizado en aplicaciones militares), HSTS [Muscettola, 1994] (utilizado en la planificación de las observaciones del telescopio espacial *Hubble*) e IxTeT [Ghallab y Laruelle, 1994; Vidal y Ghallab, 1996].

La planificación basada en grafos de planificación y en satisfacibilidad, con Graphplan [Blum y Furst, 1997] y SATPLAN [Kautz y Selman, 1992, 1996] a la cabeza respectivamente, ha revivido el interés por la planificación, en lo que algunos autores han pasado a denominar *planificación neoclásica* [Ghallab y otros, 2004]. Ambas aproximaciones han supuesto un importante punto de partida para el desarrollo de planificadores más modernos, potentes y expresivos como IPP [Köhler y otros, 1997; Nebel y otros, 1997], BlackBox [Kautz y Selman, 1998], SGP [Anderson y otros, 1998; Weld y Smith, 1998], STAN [Fox y Long, 1999], TGP [Smith y Weld, 1999] o TPSYS [Garrido y Onaindía, 2006] entre otros. Algunos de estos planificadores participaron en la primera competición internacional de planificación celebrada en 1998 [IPC, 2006]. Esta competición surgió como un punto de encuentro para comparar y mostrar los avances de las distintas técnicas y sistemas de planificación. Para facilitar esta comparación y proporcionar un lenguaje común de descripción de problemas se definió PDDL, que se ha ido extendiendo con nuevas funcionalidades [Edelkamp y Hoffmann, 2004; Fox y Long, 2003; Gerevini y Long, 2006; McDermott, 1998]. PDDL ha sido testigo de lo que probablemente han sido los años de mayores avances en el campo de la planificación, gracias a la planificación heurística revitalizada por HSP [Bonet y Geffner, 1999]. HSP planteó heurísticas como las h_{sum} y $h_{máx}$ vistas en la sección 13.6.3 y dio pie a otros planificadores como GRT [Refanidis y Vlahavas, 1999], AltAlt [Kambhampati y Sanchez Nigenda, 2000; Nguyen y otros, 2002], TP4 [Haslum, 2004; Haslum y Geffner, 2001] o Sapa [Do y Kambhampati, 2001, 2003], que proporciona características de optimización multicriterio. FF [Hoffmann, 2000; Hoffmann y Nebel, 2001], junto con su versión numérica metric-FF [Hoffmann, 2003], ha sido uno de los planificadores con más influencia en los últimos años por su gran eficiencia. FF basa sus heurísticas en planes relajados y sus ideas se utilizan en otros planificadores como Mips [Edelkamp y Helmert, 2001] o LPG [Gerevini y otros, 2003, 2004; Gerevini y Serina, 2002], basado en búsqueda local no determinista. La tendencia en los últimos años ya no es construir un planificador desde cero, sino reutilizar funcionalidades de los ya existentes como FF o LPG. En esta línea se han desarrollado Macro-FF [Botea y otros, 2004], Fast (Diagonally) Downward [Helmert y Richter, 2004], YAHSPP [Vidal, 2004], SGPlan [Chen y otros, 2004, 2005; Hsu y otros, 2006], MIPS-XXL [Edelkamp y otros, 2006], SATPLAN04 [Kautz y Selman, 2006], MaxPlan [Xing y otros, 2006], YochanPS [Benton y Kambhampati, 2006] y muchos más. De todos ellos, el más rápido en la última competición de planificación de 2006 es SGPlan, que utiliza técnicas de descomposición de problemas para reducir la complejidad del problema original.

En las últimas décadas, el campo de la planificación ha crecido tanto y tan deprisa que es difícil encontrar libros de consulta que cubran en amplitud las principales técnicas y algoritmos. Aunque los libros de Russell y Norvig [Russell y Norvig, 2004] y Nilsson [Nilsson, 2000] dedican capítulos específicos a la planificación, es el libro de Yang [Yang, 1997] el que está dedicado íntegramente a la planificación, introduciendo los algoritmos básicos, técnicas de descomposición y abstracción jerárquica. En este

momento, el libro más actualizado y completo sobre planificación es el de Ghallab *et al.* [Ghallab y otros, 2004] que hace un recorrido detallado y muy didáctico de todos los aspectos de planificación. La realización de este capítulo se ha visto influida por estos cuatro textos, y más concretamente por el último.

La planificación es un campo muy activo dentro de la IA y los artículos más novedosos se publican en las principales revistas y conferencias sobre IA. Además, los investigadores e interesados en planificación cuentan con una conferencia anual específica sobre planificación y scheduling denominada ICAPS (*International Conference on Automated Planning and Scheduling*) que abarca todos los terrenos de la planificación.

13.10 Ejercicios resueltos

13.1. El problema del *mundo de bloques* ha sido uno de los problemas más utilizados en el campo de la planificación. Dicho problema consiste en construir torres, apilando y desapilando bloques con la ayuda de un robot. En la Tabla 13.7 se muestra el dominio, utilizando 4 operadores, para este problema, así como el estado inicial y objetivo para un problema conocido como la *anomalía de Sussman*. Para resolver este problema, es necesario deshacer la torre inicial y construir una nueva que se corresponda con el objetivo (véase la Figura 13.5). A continuación abordaremos la resolución de este problema mediante las técnicas de búsqueda hacia delante, hacia atrás, basada en POP y en Graphplan.

Estado inicial (\mathcal{I})	$\text{libre}(C) \wedge \text{sobre}(C, A) \wedge \text{en-mesa}(A) \wedge \text{libre}(B) \wedge \text{en-mesa}(B) \wedge \text{rob-libre}$
Objetivo (\mathcal{G})	$\text{sobre}(A, B) \wedge \text{sobre}(B, C)$
Operadores (\mathcal{O})	<p>recoger(?)</p> <p>Pre: $\text{libre}(\text{?b}) \wedge \text{en-mesa}(\text{?b}) \wedge \text{rob-libre}$ Efe: $\text{sujeto}(\text{?b}) \wedge \neg\text{libre}(\text{?b}) \wedge \neg\text{en-mesa}(\text{?b}) \wedge \neg\text{rob-libre}$</p> <p>soltar(?)</p> <p>Pre: $\text{sujeto}(\text{?b})$ Efe: $\text{libre}(\text{?b}) \wedge \text{en-mesa}(\text{?b}) \wedge \text{rob-libre} \wedge \neg\text{sujeto}(\text{?b})$</p> <p>apilar(?,?)</p> <p>Pre: $\text{sujeto}(\text{?b1}) \wedge \text{libre}(\text{?b2})$ Efe: $\text{sobre}(\text{?b1}, \text{?b2}) \wedge \text{libre}(\text{?b1}) \wedge \text{rob-libre} \wedge \neg\text{sujeto}(\text{?b1}) \wedge \neg\text{libre}(\text{?b2})$</p> <p>desapilar(?,?)</p> <p>Pre: $\text{sobre}(\text{?b1}, \text{?b2}) \wedge \text{libre}(\text{?b1}) \wedge \text{rob-libre}$ Efe: $\text{sujeto}(\text{?b1}) \wedge \text{libre}(\text{?b2}) \wedge \neg\text{sobre}(\text{?b1}, \text{?b2}) \wedge \neg\text{rob-libre} \wedge \neg\text{libre}(\text{?b1})$</p>

Tabla 13.7: Definición de un problema sencillo con 4 operadores para construir una torre de 3 bloques.

1. **Resolución mediante una búsqueda hacia delante** El nodo raíz del árbol de la Figura 13.6 representa el estado inicial \mathcal{I} . Al expandir dicho nodo hay dos acciones aplicables (**recoger(B)** y **desapilar(C,A)**), ya que son las dos únicas acciones cuyas precondiciones se satisfacen en \mathcal{I} . Los estados resultantes son el producto de borrar los efectos negativos y añadir los efectos positivos de las acciones correspondientes. A partir del nodo 1 se escoge un nodo a expandir, por ejemplo el nodo 2, y se procede añadiendo los efectos correspondientes.

Los estados repetidos aparecen marcados con un aspa, indicando así que la expansión del árbol no debe continuar a partir de dicho nodo. Con líneas punteadas se indican los nodos que deben generarse en el siguiente nivel (ignorando los estados repetidos). Nótese también que si la acción aplicable desde un nodo es la inversa de la acción que ha generado dicho nodo (por ejemplo, a partir de los nodos 2, 3 y 4), la expansión del árbol no continúa por dicho nodo. Como ejercicio, se propone al lector completar el árbol de búsqueda hacia delante de la Figura 13.6.

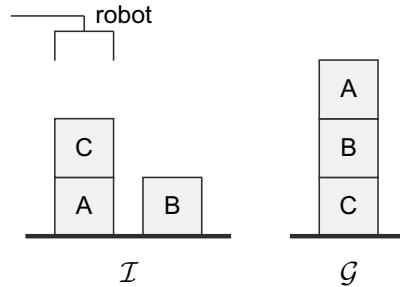


Figura 13.5: Estado inicial y objetivo del problema de la anomalía de *Sussman*.

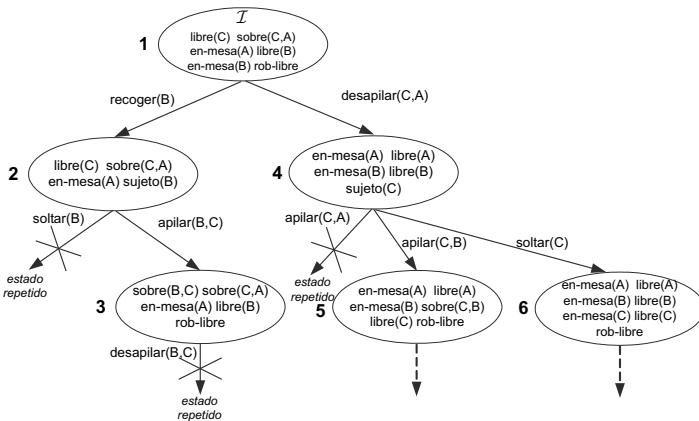


Figura 13.6: Árbol parcial generado para el problema de *Sussman* en una búsqueda hacia delante en un espacio de estados.

2. **Resolución mediante una búsqueda hacia atrás** El primer estado objetivo del problema contiene los dos literales de \mathcal{G} y es el primer nodo que se expande.

Existen dos acciones relevantes para conseguir cada uno de los literales de \mathcal{G} : `apilar(B,C)` y `apilar(A,B)`. En la Figura 13.7 se muestra un fragmento de la expansión del árbol que se origina a partir del sucesor generado con la acción `apilar(A,B)` (nodo 2). El nodo 2 será entonces el siguiente nodo que se expande, existiendo múltiples acciones para conseguir sus objetivos: 1) `apilar(B,C)` para conseguir `sobre(B,C)`, 2) `soltar(B)` y `apilar(B,?y)` para conseguir `libre(B)`, 3) `desapilar(A,?x)`, y 4) `recoger(A)` para conseguir `sujeto(A)`. Si se expande el nodo etiquetado con el número 3 se puede observar que se alcanzan estados inconsistentes; por ejemplo, la aplicación de la acción `recoger(B)` para satisfacer el literal `sujeto(B)` borraría el literal `libre(B)`, que también forma parte del estado objetivo, y por tanto produciría un estado inconsistente. Si a continuación se expande el nodo etiquetado como nodo 4, se producirá igualmente un estado inconsistente. La expansión a partir del nodo 5 produciría tres estados objetivos (sin contabilizar los estados repetidos ni inconsistentes). La solución a este problema se encuentra a través de uno de los nodos generados a partir del nodo 5. Como ejercicio, se propone al lector completar el árbol de búsqueda hacia atrás de la Figura 13.7 y se recomienda que continúe la expansión a partir de los nodos etiquetados como 6, 7 y 8.

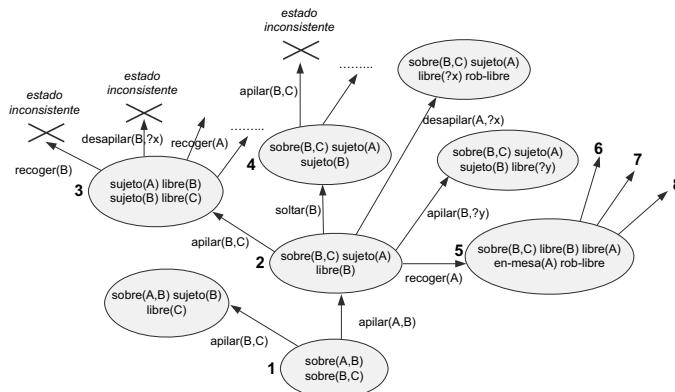


Figura 13.7: Árbol parcial generado para el problema de *Sussman* en una búsqueda hacia atrás.

■

3. **Resolución mediante planificación de orden parcial** Inicialmente (véase la Figura 13.8-a)), de los dos objetivos a resolver se selecciona uno de ellos, por ejemplo `sobre(A,B)`. Ambos objetivos tienen una única forma de resolución, mediante la introducción de un operador `apilar`, de modo que la aplicación de las heurísticas vistas en la sección 13.5.3 no permitirían discriminar entre ambos objetivos. En la Figura 13.8-b) del árbol existen tres objetivos pendientes

de resolución: `sobre(B,C)` y las dos precondiciones del operador `apilar(A,B)`. Existen dos formas de resolver el objetivo `sujeto(A)`: mediante la introducción del operador `recoger(A)` o `desapilar(A,?x)`. Por otro lado, existen tres formas de resolver `libre(B)`: mediante los operadores `soltar(B)` o `apilar(B,?x)`, o mediante el estado inicial `I`. Sin embargo, sólo existe un modo de resolver `sobre(B,C)` y, por consiguiente, escogemos dicho objetivo como el siguiente a resolver. En el plan mostrado en la Figura 13.8-c) existen cuatro precondiciones pendientes de resolver. Para las dos precondiciones del tipo `sujeto` existen dos formas de resolución, tres para el objetivo `libre(C)` y cuatro para `libre(B)`. Se deja al lector la resolución del resto del ejercicio, partiendo de la Figura 13.8-c) y averiguando cuántas formas diferentes existen de resolver los objetivos pendientes de un nodo y seleccionando aquel objetivo que se considere más apropiado.

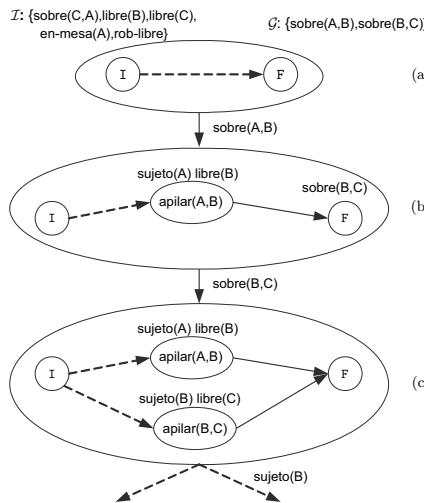


Figura 13.8: Árbol parcial generado para el problema de *Sussman* en una planificación de orden parcial.

■

4. Resolución mediante Graphplan

El primer paso de Graphplan consiste en la construcción del grafo de planificación. La Tabla 13.8 muestra los niveles de acción del grafo de planificación resultante. Aunque el nivel de acción $A_{[6]}$ coincide plenamente con $A_{[5]}$, los mutex entre `sobre(A,B)` y `sobre(B,C)` no desaparecen hasta $P_{[6]}$, nivel desde el que comienza la etapa de extracción del plan. A partir de este nivel se realiza la búsqueda regresiva tal y como se explica en el Apartado 13.6.2.2, encontrándose el plan $\langle \{\text{desapilar}(C,A)\}_{[1]}, \{\text{soltar}(C)\}_{[2]}, \{\text{recoger}(B)\}_{[3]}, \{\text{apilar}(B,C)\}_{[4]}, \{\text{recoger}(A)\}_{[5]}, \{\text{apilar}(A,B)\}_{[6]} \rangle$. Se deja como ejercicio

al lector completar el grafo de planificación representado en la Tabla 13.8 con los niveles de proposición y la etapa de extracción del plan.

$A_{[1]}$	$A_{[2]}$	$A_{[3]}$	$A_{[4]}$	$A_{[5]}$
recoger(B)	recoger(B)	recoger(B)	recoger(B)	recoger(B)
desapilar(C,A)	desapilar(C,A)	desapilar(C,A)	desapilar(C,A)	desapilar(C,A)
apilar(C,A)	apilar(C,A)	apilar(C,A)	apilar(C,A)	apilar(C,A)
apilar(C,B)	apilar(C,B)	apilar(C,B)	apilar(C,B)	apilar(C,B)
soltar(B)	soltar(B)	soltar(B)	soltar(B)	soltar(B)
soltar(C)	soltar(C)	soltar(C)	soltar(C)	soltar(C)
apilar(B,C)	apilar(B,C)	apilar(B,C)	apilar(B,C)	apilar(B,C)
	recoger(A)	recoger(A)	recoger(A)	recoger(A)
	recoger(C)	recoger(C)	recoger(C)	recoger(C)
	desapilar(C,B)	desapilar(C,B)	desapilar(C,B)	desapilar(C,B)
	desapilar(B,C)	desapilar(B,C)	desapilar(B,C)	desapilar(B,C)
		soltar(A)	soltar(A)	soltar(A)
		apilar(B,A)	apilar(B,A)	apilar(B,A)
		apilar(A,B)	apilar(A,B)	apilar(A,B)
		apilar(A,C)	apilar(A,C)	apilar(A,C)
			desapilar(B,A)	desapilar(B,A)
			desapilar(A,B)	desapilar(A,B)
			desapilar(A,C)	

Tabla 13.8: Niveles de acción del grafo de planificación de Graphplan para el problema de *Sussman*. Para simplificar, las acciones *no-op* no se muestran.

■

13.11 Ejercicios propuestos

13.1. Desarrolla el árbol de una búsqueda hacia delante para el problema de la anomalía de *Sussman* del ejercicio resuelto 1 siguiendo una estrategia de anchura y profundidad y calcula el número de nodos que es necesario expandir en cada una de las dos estrategias de búsqueda.

13.2. Si suponemos que en el problema de la anomalía de *Sussman* del ejercicio resuelto 1 existen dos robots en lugar de uno, contestar a las siguientes preguntas:

1. ¿Qué modificaciones son necesarias en la definición del problema de planificación?
2. ¿Qué estrategia de planificación habría que utilizar si se quiere encontrar la solución más corta?
3. ¿Qué diferencias habría en la resolución de este problema mediante una aproximación POP con respecto al problema original?
4. ¿Cuál sería la solución que encontraría un proceso de búsqueda hacia delante? ¿Y la encontrada mediante Graphplan?

13.3. Sea el siguiente problema de planificación donde el objetivo es intercambiar el valor de dos variables v1 y v2.

Si se ejecuta una búsqueda hacia delante para este problema, ¿cuántas iteraciones serán necesarias para encontrar la solución más corta? ¿y para la solución más larga?

Estado inicial (\mathcal{I})	$\text{valor}(v1,3) \wedge \text{valor}(v2,5) \wedge \text{valor}(v3,0)$
Objetivo (\mathcal{G})	$\text{valor}(v1,5) \wedge \text{valor}(v2,3)$
Operadores (\mathcal{O})	<p>asignar(?v,?w,?x,?y)</p> <p>Pre: $\text{valor}(?v,?x) \wedge \text{valor}(?w,?y)$ Efe: $\text{valor}(?v,?y) \wedge \neg \text{valor}(?v,?x)$</p>

Tabla 13.9: Definición de un sencillo problema de planificación para el intercambio de valor de dos variables.

13.4. Resolver el ejercicio anterior mediante una aproximación basada en POP y calcular:

1. ¿Cuántas amenazas se generan?
2. ¿Cuántos nodos se generan en el árbol POP que se desarrolla para este problema?
3. ¿Cuántos planes solución se pueden encontrar para este problema?
4. Realiza una traza en el árbol de búsqueda para encontrar la solución con el menor número de acciones.

13.5. Partiendo del problema de la anomalía de *Sussman* definido en el ejercicio resuelto 1, realizar los siguientes apartados:

1. Construir el grafo de planificación relajado (es decir, sin tener en cuenta los efectos negativos de las acciones y sin calcular las relaciones de exclusión mutua). ¿Cuál es el primer nivel en el que se inicia la etapa de extracción de un plan?
2. Comparar el tamaño (número de niveles y proposiciones/acciones en cada uno de ellos) con el del grafo de planificación generado por Graphplan en el ejercicio resuelto 1.
3. Extraer un plan relajado sobre dicho grafo de planificación. Al no contemplarse interacciones negativas, demostrar cómo este plan se puede extraer en tiempo polinómico y sin necesidad de operaciones de backtracking.
4. Construir el CSP resultante para los dos primeros niveles del grafo de planificación relajado.
5. Calcular el valor de las heurísticas h_{sum} , h_{\max} y $h_{\max 2}$ para $\mathcal{G} = \{\text{sobre}(A,B), \text{sobre}(B,C)\}$. Comparar los valores de dichas heurísticas utilizando como base de estimaciones un grafo de planificación con y sin cálculo de mutex. Comprobar también cómo varía el valor de $h_{\max 2}$ si en \mathcal{G} existen más de dos objetivos.
6. Codificar mediante fórmulas lógicas las acciones de los dos primeros niveles del grafo de planificación relajado.

13.6. Dado el grafo de planificación de la Figura 13.4 definido sobre el problema de transporte de ejemplo, continuar el proceso de extensión del mismo hasta alcanzar un nivel que sea idéntico al anterior ($A_{[t]} = A_{[t-1]}$ y $P_{[t]} = P_{[t-1]}$). ¿Cuál es este nivel y cuántas acciones/proposiciones aparecen en él? ¿Cuántas de estas acciones son del tipo no-op?

13.7. Descargar el planificador Graphplan⁷ y probarlo en distintos dominios y problemas, incluyendo el problema de transporte utilizado como ejemplo y el de la anomalía de *Sussman*. Analizar la aplicabilidad de este planificador cuando se incrementa el número de camiones y ciudades en el primer problema y el de bloques en el segundo.

13.8. Visitar la página donde se encuentran las distintas competiciones de planificación celebradas en la conferencia ICAPS⁸. Estudiar los dominios definidos en PDDL, descargar los planificadores disponibles públicamente y probar algunos de los problemas existentes.

⁷<http://www.cs.cmu.edu/~avrim/graphplan.html>

⁸<http://teamcore.usc.edu/koenig/icaps/competitions.htm>

Referencias

- ANDERSON, C.; SMITH, D.E. y WELD, D.: «Conditional Effects in Graphplan». En: *Proc. AIPS-98*, AAAI Press, 1998.
- ASUNCIÓN, M.; CASTILLO, L.; FDEZ.-OLIVARES, J.; GARCÍA-PÉREZ, O.; GONZÁLEZ, A. y PALAO, F.: «SIADERX: an Interactive Artificial Intelligence Planner for Decision Support in Forest Fire Fighting». *AI Communications*, 2005, **18**, pp. 257–268.
- BARRET, A.; CHRISTIANSON, D.; FRIEDMAN, M.; GOLDEN, K.; PENBERTHY, J.; SUN, Y. y WELD, D.: «UCPOP V4.0 User's Manual». *Informe técnico TR 93-09-06d*, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA., 1996.
- BARRET, A. y WELD, D.S.: «Partial Order Planning: Evaluating Possible Efficiency Gains». *Artificial Intelligence*, 1994, **67(1)**, pp. 71–112.
- BENTON, J. y KAMBHAMPATI, S.: «YochanPS: PDDL3 Simple Preferences as Partial Satisfaction Planning». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 23–25, 2006.
- BLUM, A.L. y FURST, M.L.: «Fast Planning through Planning Graph Analysis». *Artificial Intelligence*, 1997, **90**, pp. 281–300.
- BONET, B. y GEFFNER, H.: «Planning as Heuristic Search: New Results». En: S. Biundo y M. Fox (Eds.), *Proc. European Conference on Planning (ECP-99)*, pp. 360–372. Springer, 1999.
- BOTEA, A.; ENZENBERGER, M.; MULLER, M. y SCHAEFFER, J.: «Macro-FF». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 15–17, 2004.
- CASTILLO, L.; FDEZ.-OLIVARES, J.; GARCÍA-PÉREZ, O. y PALAO, F.: «SIADERX. Un entorno Integral de Planificación para el Diseño de Planes de Actuación en Situaciones de Crisis». En: *Jornadas de Transferencia Tecnológica de Inteligencia Artificial (TTIA-2005)*, pp. 333–342, 2005.
- CHEN, Y.; HSU, C.W. y WAH, B.W.: «SGPlan: Subgoal Partitioning and Resolution in Planning». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 30–32, 2004.
- CHEN, Y.; HSU, C.W. y WAH, B.W.: «Subgoal Partitioning and Resolution in SGPlan». En: *Proc. System Demonstration Session, Int. Conference on Automated Planning and Scheduling (ICAPS-2005)*, pp. 32–35, 2005.
- CHUN, S.B.: *Wargaming*. Air University Library. Maxwell AFB, Al., 1999.

- CURRIE, K. y TATE, A.: «O-Plan: the Open Planning Architecture». *Artificial Intelligence*, 1991, **52**(1), pp. 49–86.
- DO, M.B. y KAMBHAMPATI, S.: «Sapa: a Domain-Independent Heuristic Metric Temporal Planner». En: A. Cesta y D. Borrajo (Eds.), *Proc. European Conference on Planning (ECP-2001)*, pp. 109–120, 2001.
- DO, M.B. y KAMBHAMPATI, S.: «SAPA: a Multi-Objective Metric Temporal Planner». *Journal of Artificial Intelligence Research*, 2003, **20**, pp. 155–194.
- EDELKAMP, S. y HELMERT, M.: «The Model Checking Integrated Planning System MIPS». *AI Magazine*, 2001, pp. 67–71.
- EDELKAMP, S. y HOFFMANN, J.: «PDDL2.2: the Language for the Classical Part of IPC-4». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 2–6, 2004.
- EDELKAMP, S.; JABBAR, S. y NAZIH, M.: «Large-Scale Optimal PDDL3 Planning with MIPS-XXL». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 28–30, 2006.
- EL-KHOLY, A. y RICHARDS, B.: «Temporal and Resource Reasoning in Planning: the parcPLAN Approach». En: *Proc. 12th European Conference on AI (ECAI-96)*, pp. 614–618, 1996.
- EROL, K.; HENDLER, J. y NAU, D.S.: «HTN Planning: Complexity and Expressivity». En: *Proc. 12th Nat. Conference on AI (AAAI-94)*, pp. 1123–1128. AAAI Press, Seattle, WA, 1994.
- FIKES, R.E. y NILSSON, N.J.: «STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving». *Artificial Intelligence*, 1971, **2**, pp. 189–208.
- FINK, E. y VELOSO, M.: «Prodigy Planning Algorithm». *Informe técnico CMU-94-123*, Carnegie Mellon University, 1994.
- FOX, M. y LONG, D.: «Efficient Implementation of the Plan Graph in STAN». *Journal of Artificial Intelligence Research*, 1999, **10**, pp. 87–115.
- FOX, M. y LONG, D.: «PDDL2.1: an Extension to PDDL for Expressing Temporal Planning Domains». *Journal of Artificial Intelligence Research*, 2003, **20**, pp. 61–124.
- GARRIDO, A. y ONAINDÍA, E.: «Domain-Independent Temporal Planning in a Planning-Graph-Based Approach». *AI Communications*, 2006, **19**(4), pp. 341–367.
- GEREVINI, A. y LONG, D.: «Plan Constraints and Preferences in PDDL3». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 7–13, 2006.

- GEREVINI, A.; SAETTI, A. y SERINA, I.: «Planning Through Stochastic Local Search and Temporal Action Graphs in LPG». *Journal of Artificial Intelligence Research*, 2003, **20**, pp. 239–290.
- GEREVINI, A.; SAETTI, A.; SERINA, I. y TONINELLI, P.: «Planning in PDDL2.2 Domains with LPG-TD». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 33–34, 2004.
- GEREVINI, A. y SERINA, I.: «LPG: a Planner Based on Local Search for Planning Graphs with Action Costs». En: *Proc. 6th Int. Conference on AI Planning and Scheduling (AIPS-2002)*, pp. 281–290. AAAI Press, 2002.
- GERVASIO, M.; IBA, W.; LANGLEY, P. y SAGE, S.: «Interactive Adaptation for Crisis Response». En: *Proc. Workshop on Interactive and Collaborative Planning (AIPS-1998)*, pp. 29–36, 1998.
- GHALLAB, M. y LARUELLE, H.: «Representation and Control in IxTeT, a Temporal Planner». En: *Proc. 2nd Int. Conference on AI Planning Systems*, pp. 61–67. Hammond, 1994.
- GHALLAB, M.; NAU, D. y TRAVERSO, P.: *Automated Planning. Theory and Practice*. Morgan Kaufmann, 2004.
- GIL, Y.; DEELMAN, E.; BLYTHE, J.; KESSELMAN, C. y TANGMUNARUNKIT, H.: «Artificial Intelligence and Grids: Workflow Planning and Beyond». *IEEE Intelligent Systems*, 2004, pp. 26–33.
- GREEN, C.: «Application of Theorem Proving to Problem Solving». En: *Proc. Int. Joint Conference on AI (IJCAI-69)*, pp. 219–239. Morgan Kaufmann, San Mateo, CA, 1969.
- HASLUM, P.: «TP4'04 and HSP_a*». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 38–40, 2004.
- HASLUM, P. y GEFFNER, H.: «Heuristic Planning with Time and Resources». En: A. Cesta y D. Borrajo (Eds.), *Proc. European Conference on Planning (ECP-2001)*, pp. 121–132, 2001.
- HELMERT, M. y RICHTER, S.: «Fast Downward Making Use of Causal Dependencies in the Problem Representation». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pp. 41–43, 2004.
- HOFFMANN, J.: «A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-Climbing Algorithm». En: *Proc. 12th Int. Symp. on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, 2000.

- HOFFMANN, J.: «The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables». *Journal of Artificial Intelligence Research*, 2003, **20**, pp. 291–341.
- HOFFMANN, J. y NEBEL, B.: «The FF Planning System: Fast Plan Generation Through Heuristic Search». *Journal of Artificial Intelligence Research*, 2001, **14**, pp. 253–302.
- HSU, C-W.; WAH, B.W.; HUANG, R. y CHEN, Y.: «New Features in SGPlan for Handling Preferences and Constraints in PDDL3.0». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 39–41, 2006.
- IPC: «IPC: International Planning Competition», 2006. <Http://ipc.icaps-conference.org>.
- JÓNSSON, A.; MORRIS, P.; MUSCETTOLA, N.; RAJAN, K. y SMITH, B.: «Planning in Interplanetary Space: Theory and Practice». En: *Proc. 5th Int. Conference on AI Planning Systems (AIPS-2000)*, AAAI Press, 2000.
- KAMBHAMPATI, S. y SANCHEZ NIGENDA, R.: «Distance Based Goal Ordering Heuristics for Graphplan». En: *Proc. Int. Conference on Artificial Intelligence Planning and Scheduling*, pp. 315–332. AAAI Press, Menlo Park, CA, 2000.
- KAUTZ, H. y SELMAN, B.: «Planning as Satisfiability». En: *Proc. 10th European Conference on AI*, pp. 359–363. Wiley, Vienna, Austria, 1992.
- KAUTZ, H. y SELMAN, B.: «Pushing the Envelope: Planning, Propositional Logic and Stochastic Search». En: *Proc. 13th Nat. Conference on AI*, pp. 1194–1201, 1996.
- KAUTZ, H. y SELMAN, B.: «BLACKBOX: a New Approach to the Application of Theorem Proving to Problem Solving». En: *Proc. Workshop on Planning as Combinatorial Search*, , 1998.
- KAUTZ, H. y SELMAN, B.: «SATPLAN04: Planning as Satisfiability». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 45–46, 2006.
- KNOBLOCK, C.A: «Automatically Generating Abstraction for Planning». *Artificial Intelligence*, 1994, **68(2)**, pp. 243–302.
- KNOBLOCK, C.A.: «Deploying Information Agents on the Web». En: *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pp. 1580–1586. Morgan Kaufmann, Acapulco, Mexico, 2003.
- KÖEHLER, J.; NEBEL, B.; HOFFMANN, J. y DIMOPOULOS, Y.: «Extending Planning Graphs to an ADL Subset». En: Springer-Verlag (Ed.), *Lecture Notes in Artificial Intelligence (1348)*, pp. 273–285, 1997.

- MCALESTER, D. y ROSENBLITT, D.: «Systematic Nonlinear Planning». En: *Proc. 9th Nat. Conference on AI*, pp. 634–639. Anaheim, CA, 1991.
- MCCARTHY, J. y P.J., HAYES: «Some Philosophical Problems from the Standpoint of Artificial Intelligence». *Machine Intelligence*, 1969, 4, pp. 463–502.
- MCDERMOTT, D.: *PDDL - The Planning Domain Definition Language*. AIPS-98 Planning Competition Committee, 1998.
- MUSCETTOLA, N.: «HSTS: Integrating Planning and Scheduling». En: M. Zweben y M.S. Fox (Eds.), *Intelligent Scheduling*, pp. 169–212. Morgan Kaufmann, San Mateo, CA, 1994.
- NAREYEK, A.: «Artificial Intelligence in Computer Games - State of the Art and Future Directions». *ACM Queue*, 2004, 1(10), pp. 58–64.
- NEBEL, B.; DIMOPOULOS, Y. y KÖEHLER, J.: «Ignoring Irrelevant Facts and Operators in Plan Generation». En: *Proc. European Conference on Planning (ECP-97)*, pp. 338–350. Toulouse, France, 1997.
- NEWELL, A. y SIMON, H.: *GPS, a Program that Simulates Human Thought*. McGraw Hill, NY, 1963.
- NGUYEN, X.; KAMBHAMPTI, S. y NIGENDA, R.S.: «Planning Graph as the Basis for Deriving Heuristics for Plan Synthesis by State Space and CSP Search». *Artificial Intelligence*, 2002, 135, pp. 73–123.
- NILSSON, N.: *Inteligencia Artificial. Una Nueva Síntesis*. Mac Graw-Hill, 2000.
- PEDNAULT, E.: «ADL: Exploring the Middle Ground between Strips and the Situation Calculus». En: *Proc. Int. Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pp. 324–332. Morgan Kaufmann, San Francisco, CA, 1989.
- PENBERTHY, J.: «Planning with Continuous Change». *Informe técnico Ph.D. dissertation 93-12-01*, Dept. of Computer Science and Engineering, U. Washington, 1993. Ph.D. dissertation.
- PENBERTHY, J. y WELD, D.: «Temporal Planning with Continuous Change». *Proc. 12th Nat. Conference on AI*, 1994.
- PENBERTHY, J. y WELD, D.S.: «UCPOP: a Sound, Complete, Partial-Order Planner for ADL». En: *Proc. Int. Conference on Principles of Knowledge Representation and Reasoning*, pp. 103–114. Kaufmann, Los Altos, CA, 1992.
- REFANIDIS, I. y VLAHAVAS, I.: «GRT: a Domain Independent Heuristic for Strips Worlds based on Greedy Regression Tables». En: S. Biundo y M. Fox (Eds.), *Proc. European Conference on Planning (ECP-99)*, pp. 346–358. Springer, 1999.
- ROBOCUP RESCUE TECHNICAL COMMITTEE: «RoboCup-Rescue Simulator Manual», 2000. Available at <http://robomec.cs.kobe-u.ac.jp/robocup-rescue>.

- RUSSELL, S. y NORVIG, P.: *Inteligencia Artificial. Un Enfoque Moderno.* Pearson Educación, 2004.
- SACERDOTI, E.D.: «Planning in a Hierarchy of Abstraction Spaces». *Artificial Intelligence*, 1974, **5(2)**, pp. 115–135.
- SACERDOTI, E.D.: *A Structure for Plans and Behavior.* American Elsevier, New York, 1977.
- SMITH, D.E y WELD, D.S.: «Temporal Planning with Mutual Exclusion Reasoning». En: *Proc. 16th Int. Joint Conference on AI (IJCAI-99)*, pp. 326–337. Stockholm, Sweden, 1999.
- TATE, A.: «Generating Project Networks». En: *Proc. Int. Joint Conference on AI (IJCAI-77)*, pp. 888–893. Cambridge, MA, 1977.
- VERE, S.: «Planning in Time: Windows and Durations for Activities and Goals». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1983, **5**, pp. 246–267.
- VIDAL, T. y GHALLAB, M.: «Dealing with Uncertain Durations in Temporal Constraint Networks dedicated to Planning». En: *Proc. 12th European Conference on AI (ECAI-96)*, pp. 48–52, 1996.
- VIDAL, V.: «A Lookahead Strategy for Heuristic Search Planning». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004)*, pp. 150–159, 2004.
- WELD, A. y SMITH, D.E.: «Extending Graphplan to Handle Uncertainty and Sensing Actions». En: *Proc. AAAI-98*, pp. 897–904. Madison, WI, 1998.
- WELD, D.: «An Introduction to Least Commitment Planning». *AI Magazine*, 1994, **15(4)**, pp. 93–123.
- WILKINS, D.: *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.
- XING, Z.; CHEN, Y. y ZHANG, W.: «MaxPlan: Optimal Planning by Decomposed Satisfiability and Backward Reduction». En: *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pp. 53–55, 2006.
- YANG, Q.: *Intelligent Planning. A Decomposition and Abstraction Based Approach.* Springer-Verlag, Berlin, Heidelberg, 1997.
- YANG, Q. y TENENBERG, J.D.: «ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner». En: *Proc. American Conference on Artificial Intelligence (AAAI-90)*, pp. 204–209. AAAI Press, Boston, MA, 1990.

Capítulo 14

Control

**José M. Juárez Herrero¹, José J. Cañadas Martínez² y
Roque Marín Morales¹**
Universidad de Murcia¹ y Universidad de Almería²

14.1 Introducción

En este capítulo se estudia el control de procesos como una tarea intensiva en conocimiento. El capítulo se inicia con una introducción al problema y una revisión de conceptos básicos en control. Le sigue un planteamiento en tres niveles: a) Nivel Metodológico; b) Nivel de Técnicas y c) Nivel de Diseño.

En el primer nivel, el control se considera como una tarea genérica, en el sentido de la Ingeniería del Conocimiento, y se describen algunas propuestas de métodos de resolución de problemas para tareas de control, formuladas en el marco de la metodología CommonKADS. Se trata de métodos generales, ciertamente muy simples, que no permiten abordar con suficiente profundidad el desarrollo de sistemas inteligentes para aplicaciones prácticas en control.

Por ello, el segundo nivel se dedica a técnicas concretas, que en un marco metodológico corresponderían al nivel de inferencias. Entre las técnicas inteligentes que se han aplicado con éxito a problemas de control, tanto procedentes del paradigma simbólico como del paradigma conexiónista, se encuentran: las redes neuronales, los modelos de razonamiento cualitativo, las redes de Petri y las técnicas borrosas.

De ellas, las técnicas de control borroso han tenido un altísimo impacto en aplicaciones prácticas, y hoy en día se pueden encontrar implantadas, tanto en soluciones para el mundo industrial como en dispositivos de electrónica doméstica. Por este motivo, en este capítulo se aborda en profundidad sólo esta técnica. Cabe adelantar que las funciones de control borroso se comportan como aproximadores universales, al igual que las redes neuronales, por lo que, en teoría al menos, es posible aproximar mediante ellas cualquier función de control convencional. La gran diferencia respecto al control convencional es la posibilidad de definir estas funciones de control mediante conjuntos de reglas de producción, lo que permite modelar de forma relativamente simple el conocimiento de los expertos en supervisión y control de procesos, algo particularmente útil en aplicaciones complejas. Se obviarán los aspectos relacionados con

el aprendizaje de reglas de control fuzzy, que en gran medida se corresponden con las técnicas de aprendizaje en redes neuronales. Llegados a este punto, se tendrá una visión bastante clara de los recursos conceptuales y formales de los que se dispone hoy en día para resolver de forma inteligente problemas de control complejos.

Pero, para llegar a la implementación de una aplicación práctica, falta abordar un tercer nivel: el diseño de sistemas de control inteligente. En este tercer nivel se estudiarán arquitecturas y herramientas que soportan y facilitan esta última fase del desarrollo. Existen abundantes herramientas para el diseño e implementación de sistemas de control borroso, incluyendo algunas que facilitan las implementaciones hardware. Por su generalidad, facilidad de uso y disponibilidad, se elegirá como soporte básico MATLAB y sus librerías *Simulink* y *Fuzzy Toolbox*.

Sin embargo, la mayoría de las herramientas para control borroso no tienen en cuenta dos aspectos que son esenciales en muchas aplicaciones del control inteligente, especialmente en aquellas dirigidas al mundo industrial: el razonamiento oportunista y el razonamiento en tiempo real. La solución consiste en el uso de arquitecturas software que soporten estas características. En particular, se hará una introducción a las arquitecturas de pizarra, como una solución histórica al problema del razonamiento reactivo y oportunista, y una introducción a las herramientas para desarrollo de sistemas en tiempo real basados en conocimiento. Otras soluciones al problema del control inteligente, como los sistemas multiagentes, quedan fuera del alcance del capítulo.

14.2 Conceptos fundamentales

La ingeniería de los sistemas de control tiene como objeto el estudio, la elección y el ordenamiento de los elementos que conforman un sistema de control. En esta sección se presentan brevemente los componentes fundamentales y algunos de los tipos de control convencional más habituales.

14.2.1 Problema de control

En muchos campos de la ingeniería se estudia el diseño y mantenimiento de sistemas y procesos automáticos. Por proceso se entiende una secuencia de operaciones para obtener un fin determinado, mientras que el concepto de sistema es más general, siendo un conjunto de operadores o componentes que actúan relacionados de tal manera que realizan una tarea como una unidad completa.

Un problema habitual en la ingeniería es determinar el funcionamiento de un sistema que actúa sobre otro sistema (o proceso) con un fin concreto. Los problemas de control forman parte de estos tipos de problemas. Así, un sistema de control es un sistema que dirige o regula el comportamiento de otro sistema o proceso, formado por un conjunto de componentes capaz de gobernar una salida a partir de unos elementos de entrada.

14.2.2 Componentes fundamentales de un sistema de control

En cualquier tipo de sistema de control se pueden encontrar los siguientes elementos fundamentales:

- La **planta** es el proceso o sistema sobre el que se pretende actuar para establecer el control.
- El **controlador** gestiona el valor de la salida de control a partir de la entrada de referencia.
- La **entrada de referencia** (o entrada de mando) es la señal externa aplicada a un controlador como un parámetro de ajuste.
- La **señal de control** es la variable, manipulada por el sistema de control, aplicada a la planta.
- La **salida controlada** es la variable de salida de la planta que está siendo controlada.

14.2.3 Tipos de sistemas de control

Los sistemas de control se clasifican habitualmente según su estructura. Así, se establecen dos grandes categorías:

- Sistemas de control de lazo abierto.
- Sistemas de control de lazo cerrado.

14.2.3.1 Sistemas de control de lazo abierto

Los sistemas de control más sencillos que existen son los denominados controladores de lazo abierto. En ellos, la acción del controlador es independiente de la salida controlada de la planta. Así, los sistemas de control de lazo abierto están compuestos únicamente por el controlador, la planta, la entrada de referencia al controlador, la señal de control de la planta y la salida controlada (véase la Figura 14.1).



Figura 14.1: Controlador de lazo abierto.

En estos sistemas, para que el controlador no actúe *a ciegas*, se requiere una calibración muy precisa de la entrada de referencia, ya que no es posible (de forma automática) la corrección dinámica de la señal controlada. Sin esta calibración,

la salida controlada podría distanciarse considerablemente del valor esperado. Estos controladores están presentes en la vida cotidiana. Por ejemplo, las tostadoras tienen un sistema de control donde la calibración se establece definiendo la temperatura y el tiempo que debe permanecer la tostada. El controlador actúa sobre una resistencia, calentando el pan durante el tiempo fijado y la temperatura establecida.

14.2.3.2 Sistemas de control de lazo cerrado

Los sistemas de control de lazo abierto se comportan de forma adecuada en entornos relativamente estables. Sin embargo, cuando existen perturbaciones inesperadas, un sistema de control de lazo abierto es, a menudo, bastante ineficaz. Por lo tanto resulta necesario establecer sistemas más sofisticados.

Así, se aplica la idea de la retroalimentación para ofrecer mayor capacidad de actuación a los controladores. Los controladores que utilizan las propiedades de la retroalimentación de sistemas se denominan controladores de lazo cerrado. La retroalimentación en estos sistemas es la característica que permite que la salida controlada se compare con la entrada de referencia, de forma que el controlador actúe de forma apropiada considerando la diferencia producida entre ambos (error). Es decir, el control en los sistemas de lazo cerrado depende tanto de la entrada como del valor de salida, siendo su objetivo la minimización del error.

Los elementos fundamentales de un controlador de lazo cerrado son (véase la Figura 14.2): la entrada de referencia, la señal de error, el comparador, el controlador, la señal de control, la planta, la salida controlada, el retroalimentador y la señal de retroalimentación.

- El **comparador** (o unidad comparadora) es uno de los componentes fundamentales que diferencia a un controlador de lazo cerrado de uno de lazo abierto. El comparador proporciona una señal de salida (señal de error), que es la desviación existente entre la entrada de referencia y la señal retroalimentada. Normalmente por desviación se entiende la diferencia entre los valores de ambas señales por cada instante de tiempo.
- La **entrada de referencia**, al igual que en los sistemas de lazo abierto, es una entrada externa que sirve para establecer el valor esperado de la señal controlada, por lo que también se denomina punto de consigna.
- La **salida controlada** es una señal de salida que, a diferencia de los controladores de lazo abierto, también sirve para calibrar el controlador.
- El **retroalimentador** (o unidad de retroalimentación) es el conjunto de elementos que adquieren la salida controlada y la tratan de forma apropiada para que sea comparada con la entrada de referencia. Se entiende por **señal de retroalimentación** la obtenida a la salida del retroalimentador.

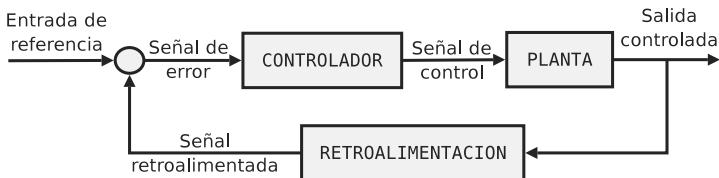


Figura 14.2: Controlador de lazo cerrado.

Los sistemas de lazo cerrado, en comparación con los de lazo abierto son, a menudo, más exactos y menos sensibles a las perturbaciones del exterior. Sin embargo, hay que tener en cuenta que estos sistemas pueden producir cierta inestabilidad en la planta debido a su continuo reajuste. Además, son sistemas más complejos que los de lazo abierto, por lo que los costes de mantenimiento son mayores.

14.2.4 Soluciones convencionales

14.2.4.1 Estructuras de los sistemas de control convencionales

Dentro de los sistemas de control convencionales, existen numerosos tipos de controladores según el funcionamiento y estructura. Entre los esquemas de control convencionales de lazo cerrado más comunes encontramos:

- Los **servosistemas reguladores** son aquellos que permiten que la salida controlada se mantenga constante frente a cambios exteriores. Por ejemplo, el sistema de climatización de un automóvil, que mantiene la temperatura interior del vehículo a una temperatura especificada.
- Los **servosistemas de posición** permiten que la salida controlada siga de forma continua los valores establecidos por la entrada de referencia. Por ejemplo, los sistemas para el control de trayectorias.
- Los sistemas de **control modelo-referencia** son esquemas de control no lineal, donde a partir de la entrada se utiliza un modelo matemático para simular el comportamiento de la planta. Así, el resultado de la simulación a partir de la entrada se compara con la salida de la planta. La diferencia entre ambas salidas es utilizada para generar la señal de control.
- Los sistemas de **control adaptativo** no sólo consideran la salida controlada, sino un conjunto de características dinámicas de la planta que afectan a unos parámetros configurables del controlador. Las variables de la planta son analizadas y, mediante un proceso de decisión, se establecen las modificaciones de los parámetros del módulo controlador con el fin de obtener un funcionamiento óptimo.

14.2.4.2 Función de transferencia

A grandes rasgos, se entiende por *transformación* a la sustitución de una función por otra en una ecuación. Por ejemplo, la derivación es la sustitución de la función $f(x)$ por su función derivada $f'(x)$ que se describe como $D[f(x)] = f'(x)$. Si además verifica que la transformada de la combinación lineal de dos funciones es igual a la combinación lineal de sus transformadas, es decir, $T[\alpha f(x) + \beta g(x)] = \alpha T[f(x)] + \beta T[g(x)]$, se dice que la *transformación es lineal*.

Una clase de transformación lineal de especial relevancia es la de las *transformaciones integrales*. En particular, la más aplicada en el campo de los sistemas dinámicos es la que propuso Pierre Simon Laplace, conocida como *transformación de Laplace* (\mathcal{L}), que se define así:

$$F(s) = \mathcal{L}(f(t)) = \int_0^{\infty} e^{-st} f(t) dt \quad (14.1)$$

Esta integral transformada tiene la capacidad de transformar la integración y derivación en multiplicación y división. Así, este operador transforma las ecuaciones diferenciales e integrales en ecuaciones polinómicas, mucho más fáciles de resolver (para más detalles véase [Puig Adam, 1964; Simmons, 1993]).

Una *Función de Transferencia* es una expresión matemática que describe el comportamiento de un sistema (o proceso) en función de la entrada e y la salida s . La expresión general de una función de transferencia (F) se expresa así:

$$F = s/e \quad (14.2)$$

En sistemas dinámicos, resulta habitual representar de forma explícita la relación entre entrada y salida en función del tiempo (t).

$$F(t) = s(t)/e(t) \quad (14.3)$$

Supóngase un sistema simple, con una sola entrada $u(t)$ y una sola salida $y(t)$. En la ingeniería, muchos procesos de comportamiento dinámico pueden ser descritos mediante ecuaciones diferenciales lineales como la que sigue:

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_1 \frac{dy}{dt} + a_0 y = b_m \frac{du^m}{dt^m} + \dots + b_1 \frac{du}{dt} + b_0 u \quad (14.4)$$

La utilización de la transformada de Laplace hace factible una representación más sencilla de la dinámica del proceso. Así, aplicando esta transformada a ambos miembros, se obtendrá:

$$G(s) = \frac{\mathcal{L}(y(t))}{\mathcal{L}(u(t))} = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + a_1 s + a_0} \quad (14.5)$$

donde G es la función de transferencia del sistema, pero expresada en el dominio transformado, en lugar de en el dominio del tiempo. Como puede observarse, la utilización

de la transformada de Laplace es muy ventajosa, puesto que simplifica considerablemente la representación de un sistema dinámico [Simmons, 1993], como en el siguiente ejemplo:

$$u(t) = Ri(t) + L(di(t)dt) + \frac{1}{C} \int i(t)dt \quad (14.6)$$

$$U(s) = RI(s) + LS(s) + \frac{1}{Cs} I(s) \quad (14.7)$$

donde con minúsculas se representan las variables del dominio temporal (como $i(t)$) y en mayúsculas las variables transformadas en el dominio de Laplace (como $I(s)$).

Por tanto, la función de transferencia $G(s)$ es la expresión que relaciona la señal de salida de un componente con la señal de entrada en el dominio de Laplace, siendo $G(s) = \mathcal{L}(s(t))/\mathcal{L}(e(t)) = S(s)/E(s)$.

Los ejemplos más comunes de la utilización de las funciones de transferencia se pueden encontrar en estos tres tipos de controladores: el controlador proporcional, el controlador integral y el controlador derivativo.

- En un **controlador proporcional**, la salida (señal de control s) es directamente proporcional a la entrada (señal de error e): $s = k_p * e$, donde k_p es una constante que se denomina ganancia proporcional. Así, la señal de control depende únicamente del error en cada instante del tiempo. Por lo tanto, la función de transferencia $G(s) = S(s)/E(s) = k_p$. De esta forma, el controlador proporcional funciona como un amplificador del valor de la señal de error. Así, los valores de señal de error grandes producen valores de salidas de control grandes. Por ejemplo, considérese un calentador de agua con un controlador proporcional en el que la señal de entrada de referencia se establece a 20°C y k_p se establece a un factor proporcional de 10. Así, cuando el valor actual es de 10°C (hay un error de 10°C), la salida de control marcaría 100 % (aumentar un 100 % el calor suministrado), cuando la temperatura es 20°C (hay un error de 0°C), la salida marcaría 0 % (no aumentar), y con 15°C (un error de 5°C) la salida marca 50 % (aumentar un 50 %).
- En un **controlador integral**, la salida es proporcional a la integral de la entrada en el tiempo. Es decir, $s = k_i \int_0^t e dt$, siendo k_i una constante que se denomina ganancia integral. La integral entre 0 y t de e es el área de la función e entre 0 y t . Así, la salida del controlador (señal de error) es proporcional a la acumulación de los errores de los instantes de tiempo anteriores. Por tanto, la función de transferencia será: $G(s) = S(s)/E(s) = k_i/s$, puesto que $S(s) = k_i E(s)/s$. De esta manera, en los controladores integrales, la salida depende de la señal de error y del tiempo durante el que la señal de salida mantiene la desviación respecto a la entrada de referencia.
- En un **control derivativo**, la salida es proporcional a la derivada de la entrada. Es decir, es proporcional a la razón de cambio de la señal de error: $s = k_d de/dt$, donde k_d es una constante que se denomina ganancia derivativa. Por tanto, la función de transferencia es: $G(s) = S(s)/E(s) = k_d s$, ya que $S(s) = k_d s E(s)$.

Así, en los controladores derivativos, la salida depende proporcionalmente de las variaciones de la señal retroalimentada. Este tipo de controladores permiten realizar acciones correctivas anticipándose a errores grandes, puesto que el control es dependiente de la variación del error. Sin embargo, cuando la variación es constante, no es posible realizar una actuación correctiva, por lo que es necesario combinar este tipo de controlador con alguno de los anteriores.

14.2.4.3 Controlador PID

A pesar de su sencillez, los controladores PID son los controladores más utilizados en la mayoría de problemas de control de procesos en la industria y en el control de procesos químicos. Los controladores PID son controladores de lazo cerrado donde la señal de control depende de la señal de error en función de su comportamiento histórico y de la tasa de variación de la señal. Esta estrategia permite al controlador PID realizar un control más estable y preciso.

La característica principal de un controlador PID es la forma en la que se calcula la señal de control, estableciendo tres medidas para procesar la señal de salida controlada: anular el error del instante actual, medir la cantidad de tiempo que la señal se ha mantenido con valores incorrectos e intentar anticipar errores futuros midiendo la tasa de variación de la señal en cada instante. A estos controladores se les denomina PID debido al tipo de cálculo que se realiza para la corrección de errores: Producto, Integración y Derivación (véase la Figura 14.3).

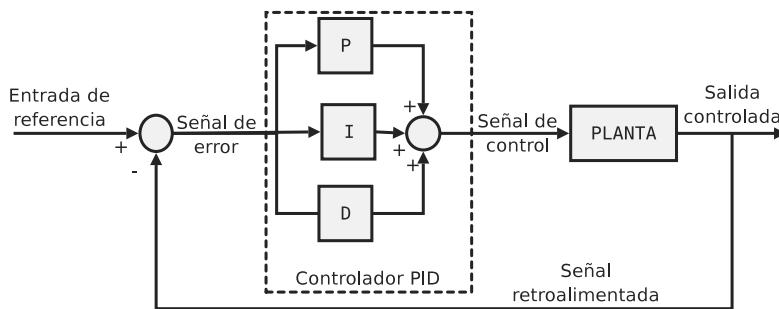


Figura 14.3: Controlador PID

Módulo Proporcional (P). Para tener en cuenta el error actual, la señal de error se multiplica por una constante para amplificar su valor y así obtener una componente de la señal de error que actúe de forma proporcional al error. El comportamiento del módulo proporcional se puede describir mediante la función de transferencia que se explicó en el apartado anterior.

Módulo Integral(I). Para tener en cuenta los valores de error ocurridos en el pasado, se realiza una operación de integración en el intervalo de tiempo determinado,

es decir, se calcula la tasa de los valores de error medidos en el pasado. Este valor se suma al valor del módulo proporcional, evitando grandes oscilaciones debidas a picos en breves períodos de tiempo de la señal de error. El comportamiento del módulo integral se puede describir mediante la función de transferencia que se explicó en el apartado anterior.

Módulo Derivativo(D). Con el fin de prevenir errores futuros, este módulo calcula la primera derivada de la señal de error. Este valor se considera como una medida de la respuesta del sistema a los cambios. Así, cuanto mayor es la derivada, más rápido responde el controlador a los cambios del sistema. El comportamiento del módulo derivativo se puede describir mediante la función de transferencia que se explicó en el apartado anterior.

En la actualidad, la mayoría de controladores simples PID que se comercializan, se desarrollan sobre microprocesadores (controladores digitales). Un controlador digital funciona de forma discreta, muestreando la señal analógica de entrada mediante dispositivos sensores. En dichos controladores, los algoritmos de control se implementan directamente en el lenguaje del microprocesador, distribuidos en bloques de cálculo y almacenados en la memoria del dispositivo controlador. Uno de los ejemplos de uso habituales de estos controladores en la industria es su utilización para el control de velocidad en automóviles.

A pesar del gran éxito de los controladores PID, también tienen sus límites. En primer lugar, están diseñados para resolver problemas de control lineal, es decir, aquellos que permiten la resolución de ecuaciones diferenciales mediante transformadas de Laplace. Además, existen situaciones, como la denominada *saturación de acción integral*, en la que la combinación de la acción integral del controlador junto con la aportación proporcional podría llevar a una situación fuera de control que debe ser subsanada mediante intervención manual (para más detalles véase [Smith y Corripio, 1994]).

14.3 Métodos genéricos para control

En esta sección se considera al control como una tarea genérica, en el sentido de la Ingeniería del Conocimiento, y se describen algunas propuestas de métodos de resolución de problemas para tareas de control en el marco de la metodología CommonKADS. Se trata de métodos generales, ciertamente muy simples, demasiado esquemáticos para abordar en profundidad el desarrollo práctico de sistemas de control inteligentes.

14.3.1 Control convencional frente a control inteligente

La estructura de un sistema de control convencional (véase la Figura 14.4) es insuficiente para el control de procesos complejos, debido a los siguientes factores:

- Mal comportamiento de este tipo de procesos frente a controladores convencionales, debido entre otros motivos, a sistemas mal definidos, la disponibilidad de

representación analítica, la existencia de incertidumbre e imprecisión, la necesidad de gestionar conceptos no numéricos, etc.

- Necesidad de aumentar la seguridad de funcionamiento, por exigencia de mayor fiabilidad (por ejemplo, centrales nucleares), o por exigencia de funcionamiento continuo (por ejemplo, altos hornos, hornos de cemento, ...).

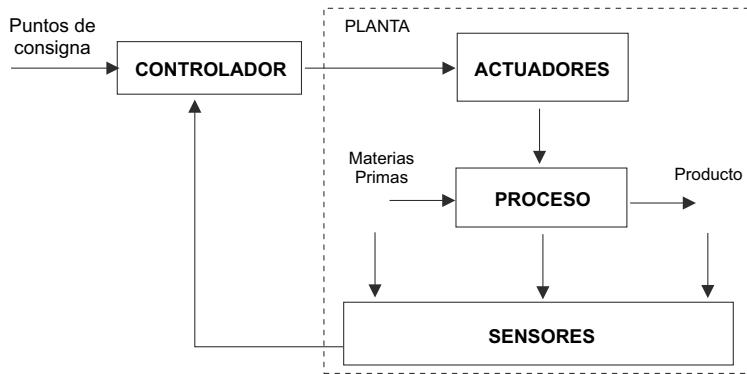


Figura 14.4: Esquema de un sistema de control convencional.

El *Control Supervisor* supone, respecto al control convencional, la introducción del operador dentro del sistema de control, generalmente como un lazo de control superpuesto al lazo de control convencional. Las tareas que se realizan en control supervisor (véase la Figura 14.5) son de dos tipos. En primer lugar, están las relacionadas con la interacción hombre-máquina, distinguiendo entre:

- La *monitorización*, que consiste básicamente en la presentación de datos de los sensores al operador y el disparo de alarmas en el caso de que algún valor medido esté fuera del rango válido.
- La interpretación de comandos de control, que consiste en la traducción de las órdenes del operador en nuevos puntos de consigna a establecer.

En segundo lugar, están las relacionadas con la toma de decisiones de control realizada por el operador, llevando a cabo la selección y aplicación de estrategias de control. Esta toma de decisiones realizada por el operador constituye el elemento más complejo de modelar en control inteligente, ya que su realización implica la necesidad de utilizar conocimiento: experiencia, juicio y pericia humanas.

Las tareas de Monitorización y Supervisión por parte del operador se realizan en una sala de control con un software integrado de tipo SCADA (Supervisory Control and Data Acquisition). Los sistemas SCADA permiten el registro de datos del proceso y la interacción de una forma cómoda a través de interfaces gráficos. Facilitan la presentación y almacenamiento de la información, pero continúa siendo el operador de la sala de control el que decide sobre la evolución del proceso, a la vista de los datos presentados.

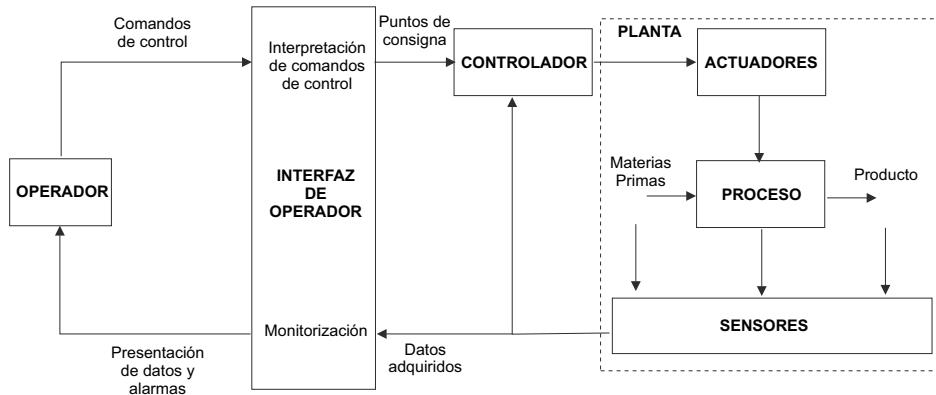


Figura 14.5: Esquema de un sistema de control supervisor.

El control inteligente (véase la Figura 14.6) surge como soporte para la sistematización de las tareas complejas realizadas por el operador, con el fin de suplir sus posibles carencias producidas, entre otros motivos, por la sobrecarga de información o la fatiga. Se trata de automatizar, en la medida en que sea posible, tareas como el análisis de los datos, la detección de fallos, el diagnóstico de los mismos y la toma de decisiones o propuesta de acciones concretas [Colomer y otros, 2000].

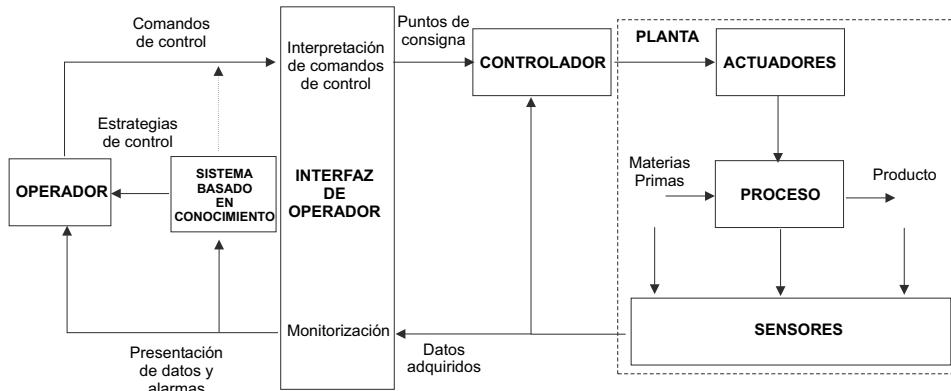


Figura 14.6: Esquema de un sistema de control inteligente.

14.3.2 Tarea genérica para el Control

Existe un vacío en CommonKADS respecto al control inteligente ya que no hay una tarea básica para el Control. La tarea genérica más parecida es *Remedio* o *Reparación*, un subtipo de *Modificación*, cuyo objetivo es restaurar la funcionalidad de un sistema, deteniendo un proceso defectuoso que la ha alterado. Sin embargo, *Modificación* no

existe como tarea básica en las últimas versiones de CommonKADS. Por tanto, es necesario construir una plantilla de tarea para *Control Inteligente*, entendida como una tarea de supervisión inteligente, obtenida por combinación de otras tareas.

Como hemos indicado al definir el control supervisor, éste se basa en tareas como *Monitorización*, realizada mediante el sistema SCADA, y otras como *Valoración*, *Diagnóstico* y *Planificación*, realizadas por el operador a la vista de los datos y alarmas proporcionados por la interfaz del sistema SCADA. A continuación se construye un método para control inteligente, basado en las plantillas de estas tareas genéricas.

Un sistema de *Monitorización* adquiere y procesa un conjunto de señales provenientes de un entorno y genera descripciones de alto nivel de abstracción sobre los objetos y eventos de dicho entorno. El objetivo de la correspondiente tarea es analizar un proceso en curso para detectar si se comporta de acuerdo con las expectativas. La Figura 14.7 muestra la plantilla de la tarea Monitorización, extendida para problemas de control.

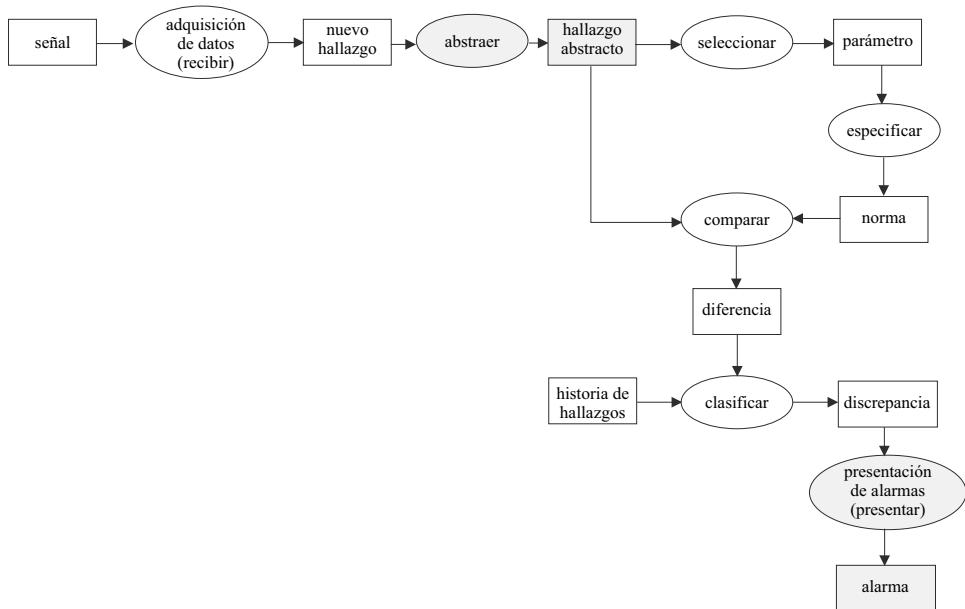


Figura 14.7: Plantilla para Monitorización.

La *Valoración* tiene por objetivo establecer una categoría de decisión para un caso, basándose en normas específicas del dominio. Su plantilla se muestra en la Figura 14.8.

El objetivo de la tarea de *Planificación* es generar un plan, formado por un conjunto de acciones, para alcanzar un objetivo. CommonKADS no proporciona un método específico, sino que recomienda usar métodos de síntesis o de diseño por configuración, según que el espacio de planes sea reducido o grande, respectivamente (véase la Figura 14.8).

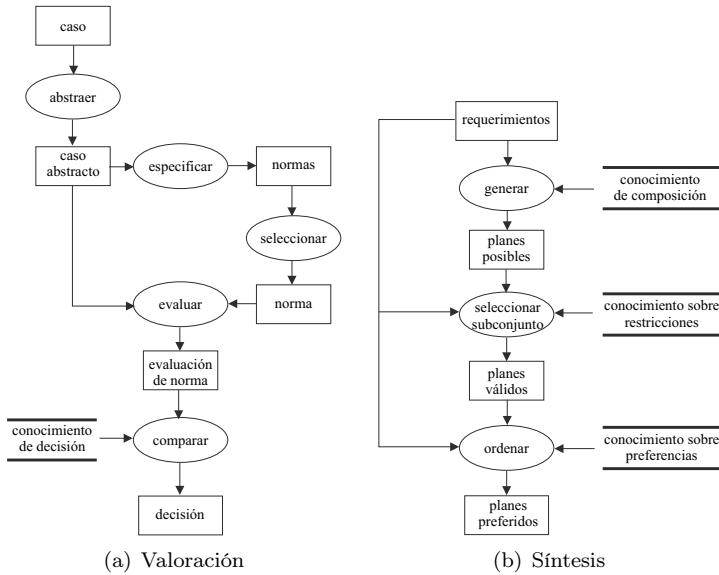


Figura 14.8: Plantillas para Valoración y Síntesis.

Por último, el objetivo del *Diagnóstico* (véase la Figura 14.9) consiste en encontrar un factor causal que explique un comportamiento observado discrepante respecto a la norma o a las expectativas. La Figura 14.10 muestra un método para *Control Inteligente*, obtenido como combinación de las plantillas de tareas comentadas previamente, y constituye un punto de partida para abordar el diseño de un sistema de control inteligente.

14.4 Control borroso

Tradicionalmente, los sistemas de control inteligente combinan la teoría clásica de control con técnicas de IA para obtener un sistema basado en un modelo físico, o matemático, dirigido por cierto conocimiento adicional. El estudio de estos sistemas se ha apoyado en campos de la IA tales como las redes neuronales, los algoritmos genéticos, los sistemas multiagente o la lógica borrosa.

En este apartado se estudiarán los sistemas de control inteligentes de mayor implantación, los controladores borrosos. El objetivo de estos controladores es intentar compensar las discrepancias entre los valores observados y los esperados, utilizando ciertas reglas de actuación, especificadas mediante descriptores lingüísticos. Por ejemplo, la utilización de términos tales como *muy caluroso* o *poco rápido* representan información cualitativa poco precisa, pero son un elemento sustancial del conocimiento humano. Estos sistemas son fáciles de entender por el experto y su implementación es sencilla. Además, debido a que su manufacturación es relativamente económica, estos controladores inteligentes se han incorporado velozmente al mundo de la industria; en

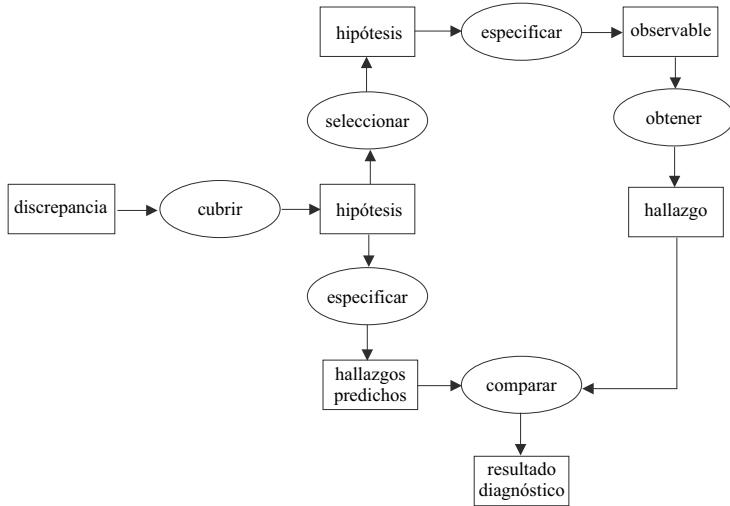


Figura 14.9: Plantilla para Diagnóstico.

ocasiones, combinándolos con controladores PID tradicionales. Algunas aplicaciones del uso de controladores borrosos son: hornos de cemento, trenes y suburbanos (Hitachi, control del metro de Sendai, Japón), vuelo sin piloto, robots móviles (NASA), control de purificación de agua (Fuji Elec. y TIT), centro de suministro doméstico de agua (Matsushita), control de grupos de ascensores (Mitsubishi Elec.), cámaras digitales (enfoque, estabilización de imagen), lavadoras... etc.

14.4.1 Estructura básica

Un sistema de lógica borrosa puro está formado únicamente por dos componentes:

- Una base de reglas borrosas.
- Un mecanismo de inferencias borrosas.

La base de reglas es un conjunto de reglas borrosas (SI-ENTONCES) expresadas en forma lingüística. Estas reglas se describen mediante una expresión que contiene un conjunto de términos borrosos en la parte del antecedente (SI) y otro conjunto de términos borrosos en la parte del consecuente (ENTONCES) (véase el capítulo 3 sobre Sistemas Basados en Reglas). En control borroso, se les denomina términos de la parte izquierda o LHS (del inglés Left Hand Side), y términos de la parte derecha o RHS (del inglés Right Hand Side), respectivamente. Por ejemplo, en el campo de la domótica, un controlador de temperatura podría utilizar la regla: *SI la humedad de la habitación es alta y la temperatura se incrementa rápidamente, ENTONCES activar el aire acondicionado a una potencia media-alta*.

El mecanismo de inferencias borrosas busca en la base de reglas borrosas las que son aplicables a la situación actual, y opera con ellas de forma que el espacio de

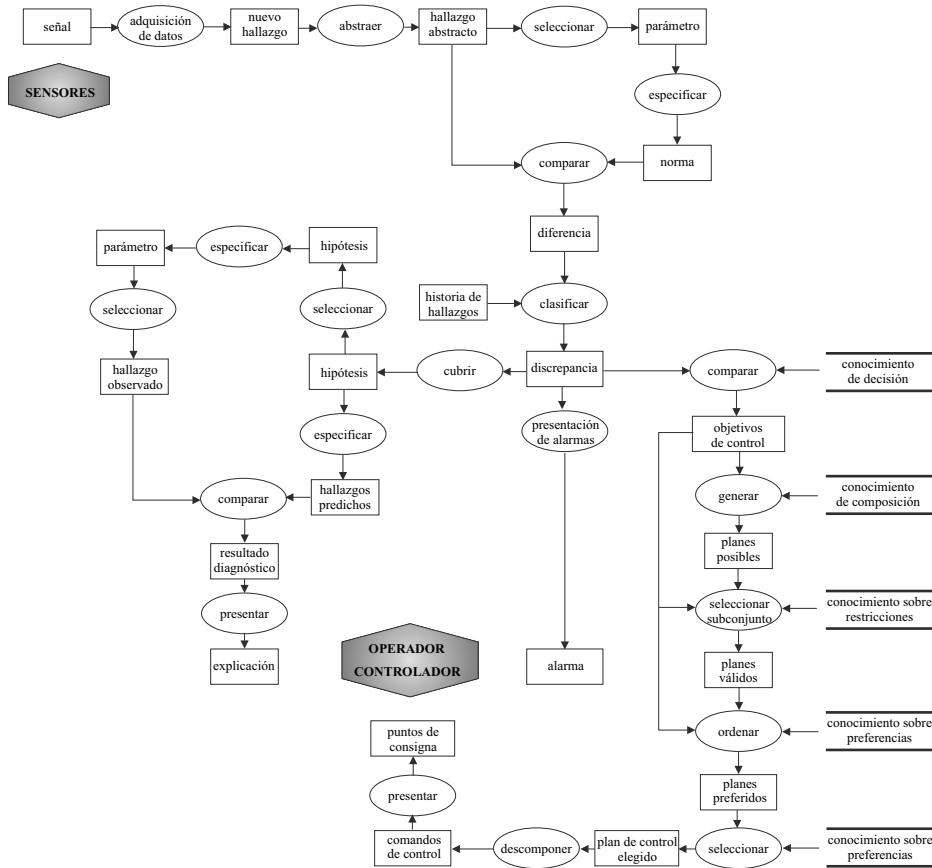


Figura 14.10: Plantilla para Control Inteligente.

entradas sea proyectado en el espacio de salidas. Sin embargo, es habitual que las entradas y las salidas de los controladores sean números reales precisos. Para poder utilizar reglas borrosas a partir de entradas y salidas concretas, existen principalmente dos soluciones:

- Sistema borroso de Takagi-Sugeno.
- Utilización de fuzzificador/defuzzificador.

El sistema borroso de Takagi-Sugeno establece las entradas y salidas del controlador como variables con valores reales, operando directamente con ellos. Una solución alternativa consiste en añadir un componente de fuzzificación y otro de defuzzificación al sistema. En esta configuración, se interpone un fuzzificador en la entrada del mecanismo de inferencia, que transforma la entrada numérica precisa en un conjunto borroso. Además, se interpone un defuzzificador en la salida, que traduce los conjuntos borrosos resultantes de aplicar las reglas a una salida numérica precisa.

Así, los sistemas borrosos se clasifican en tres tipos según su estructura:

- Sistema borroso puro.
- Sistema borroso de Takagi-Sugeno.
- Sistema borroso con fuzzificador/defuzzificador.

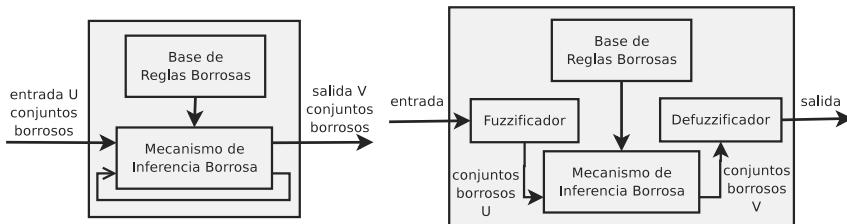


Figura 14.11: Sistema de lógica borrosa puro (izquierda); sistema de lógica borroso con fuzzificador y defuzzificador (derecha).

Independientemente de la arquitectura utilizada, una cuestión clave es la obtención de las reglas borrosas SI-ENTONCES que se utilizan en el controlador. Existen dos formas de obtener dichas reglas. La primera consiste en que el experto del dominio de aplicación describa dichas reglas. Para ello, resulta conveniente la utilización de técnicas de adquisición de conocimiento (véase el capítulo 3). La segunda consiste en aplicar algoritmos de aprendizaje y utilizar los datos obtenidos para generar y ajustar las reglas (véase el capítulo 18 sobre Extracción de Reglas). Así, los sistemas borrosos se clasifican en dos tipos según el origen de las reglas borrosas SI-ENTONCES:

- Sistemas estáticos (experto).
- Sistemas adaptativos (algoritmos de aprendizaje).

Existen varias alternativas a la hora de configurar los sistemas de control borrosos. Sin embargo, la configuración más habitual es un sistema estático donde se interpone un fuzzificador en las entradas y un defuzzificador en las salidas. Además, asumiremos que estos controladores operan en instantes de tiempos discretos, por lo que para cada instante de tiempo $t = t_k$ el controlador aplica alguna de las reglas para obtener una salida. La frecuencia de muestreo de las entradas dependerá del tipo de proceso a controlar. Así, para el desarrollo de un controlador borroso de este tipo, tendremos que especificar:

1. Variables de entrada LHS y de salida RHS de las reglas.
2. Método de fuzzificación.
3. Base de reglas borrosas.

4. Mecanismo de inferencias.

5. Método de defuzzificación.

En el resto de esta sección analizaremos en detalle cada uno de estos pasos, en los que se desglosa el desarrollo de un sistema de control borroso estático con fuzzificador/defuzzificador.

14.4.2 Variables LHS / RHS

Si se desean utilizar reglas borrosas SI-ENTONCES en un sistema de control, es necesario establecer qué términos lingüísticos van a ser utilizados. Para ello, hay que dividir el universo de discurso en un número finito de conjuntos borrosos, dándole una etiqueta lingüística a cada uno de ellos. Es decir, para cada una de las variables precisas, tanto de la entrada como de la salida, hay que establecer cuál es el rango de valores que definen cada uno de los conjuntos borrosos etiquetados. Así, las variables LHS estarán formadas por los conjuntos borrosos definidos sobre las variables de entrada, y las variables RHS estarán formadas por los conjuntos borrosos definidos sobre las variables de salida.

Como vimos en el capítulo 7 , para establecer estas etiquetas lingüísticas, hay que elegir las funciones de pertenencia mediante las que se definen los correspondientes conjuntos borrosos. Habitualmente estas funciones son descritas a mano, por lo que se suelen utilizar funciones relativamente sencillas como: triangulares, rectangulares, trapezoidales o gaussianas.

Sin embargo, un número excesivo de conjuntos borrosos podría complicar demasiado la legibilidad de las reglas borrosas. Para evitar este problema, las variables LHS y RHS se suelen:

- Escalar a un mismo universo de discurso.
- Dividir en un mismo conjunto de valores borrosos.

Por ejemplo, supongamos un sistema de control con dos variables de entrada (x, z) y una salida (y) donde las variables de entrada son valores concretos entre 1 y 10 y la salida de control es un valor entre 4 y 13. Podríamos definir dos conjuntos fuzzy para establecer cuándo x o z son valores *bajo*, *bajo-medio* o *medio-alto* arbitrariamente mediante tres conjuntos borrosos (A, B y C). Del mismo modo, podríamos establecer un conjunto de etiquetas lingüísticas para describir la salida, por ejemplo valores *bajos*, *medios*, *altos* y *muy altos*, estableciendo en el rango de valores de la salida (4 a 13) más conjuntos borrosos (B,C,D y F). Como el universo de discurso es común, podría compartirse el mismo conjunto de valores borrosos para la entrada y en la salida, como se describe en la Figura 14.12.

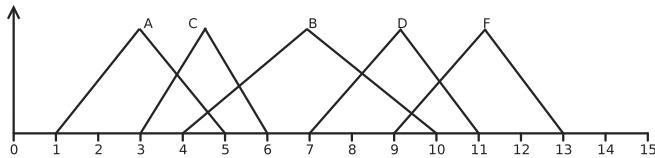


Figura 14.12: Variables borrosas sobre un mismo universo de discurso.

14.4.3 Obtención de entradas borrosas: fuzzificación

En general, la fuzzificación es el proceso mediante el cual se transforma un valor numérico concreto de una variable de entrada en un conjunto borroso que representa dicho valor de forma imprecisa.

Consideremos un conjunto de variables de entrada (x_1, \dots, x_n) , representada en forma de un vector columna $x = (x_1, \dots, x_n)^t$. Un fuzzificador proyecta un punto preciso $x' \in U = U_1 \times \dots \times U_n$ en un conjunto borroso A' del conjunto U (universo de discurso). Los fuzzificadores se clasifican en dos categorías: unitarios y no unitarios.

Un fuzzificador unitario produce un conjunto borroso unitario, es decir, un conjunto cuyo soporte se reduce a un punto, el correspondiente al valor preciso de la entrada x_p , que tiene grado de pertenencia máximo. Los restantes puntos del universo no pertenecen al conjunto en ningún grado. Para ello, se utiliza una función de pertenencia bivaluada:

$$\mu_A(x) = \begin{cases} 1 & x \text{ es } x = x_p \in U \\ 0 & x \text{ es } x \neq x_p \in U \end{cases} \quad (14.8)$$

Un fuzzificador no unitario produce un conjunto borroso cuyo soporte contiene más de un punto. En estos fuzzificadores, el valor de pertenencia es máximo $\mu(x) = 1$ para $x = x_p$. Conforme x se aleja de x_p , el valor de pertenencia decrece. Algunos ejemplos de funciones de pertenencia habituales son las triangulares, trapezoidales, o gaussianas.

14.4.4 Base de Reglas Borrosas

La base de reglas borrosas es la parte de la arquitectura del sistema de control borroso donde se almacena el conocimiento en forma de reglas SI-ENTONCES, definidas por los conjuntos borrosos LHS y RHS. Es decir, la base de reglas borrosas es un conjunto de reglas de la forma:

$$R^l : SI x_1 \text{ es } F_i^l \wedge \dots \wedge x_n \text{ es } F_n^l ENTENCES y \text{ es } G^l \quad (14.9)$$

donde R^l es la l -ésima regla de la base de reglas, F_i^l es un conjunto borroso en el universo U_i , G^l es un conjunto borroso en el universo V , y donde x_i e y son variables lingüísticas. Los conjuntos borrosos F_i^l y G^l corresponden a alguno de los términos lingüísticos definidos para las variables LHS y RHS, respectivamente. Además, llamaremos M al número de reglas borrosas en la base de conocimiento, por lo que $1 \leq l \leq M$.

En este capítulo consideraremos sistemas donde las reglas tienen múltiples entradas y una única salida, como en la expresión anterior. Un sistema con múltiples salidas siempre puede ser reducido a una combinación de sistemas con única salida. La Figura 14.13 muestra un ejemplo de base de reglas borrosas para un sistema con dos entradas, x y z , y una única salida y .

R1:SI x es A ENTONCES y es B
R2:SI z es C ENTONCES y es D
R3:SI x es A Y z es C ENTONCES y es F
R4:SI x es B ENTONCES y es F
R5:SI x es C ENTONCES y es D
R6:SI z es A ENTONCES y es B

Figura 14.13: Base de reglas borrosas.

14.4.5 Mecanismo de inferencias

En cada instante discreto, el mecanismo de inferencias puede activar una, varias o ninguna regla de la base de reglas. La activación de una regla significa que un conjunto borroso de entrada definido sobre $U = U_1 \times \dots \times U_n$ (la entrada fuzzificada) es proyectado en un conjunto borroso de salida definido sobre V . Cada regla se activará en un grado distinto, dependiendo del grado en el que se satisfagan los antecedentes de la misma.

Durante el desarrollo de un sistema borroso, para establecer el mecanismo de inferencias de reglas, hay que seleccionar:

1. Una interpretación de la implicación en las reglas SI-ENTONCES.
2. Una regla de inferencia GMP.
3. Un operador de agregación para componer los resultados de las distintas reglas.

14.4.5.1 Implicación de reglas

Una implicación borrosa ($A \rightarrow B$) se define como un tipo especial de relación borrosa en $X \times Y$, definida mediante alguna función de pertenencia particular $\mu_{A \rightarrow B}(x, y)$ (véase el capítulo 3). En la Tabla 14.1 se describen los operadores de implicación más utilizados en el campo del control borroso.

Los operadores de implicación deben ser elegidos convenientemente, atendiendo a criterios tales como requisitos del dominio o características físicas. Por ejemplo, la implicación de Mamdani (mínimo) se utiliza con frecuencia, debido a lo sencillo de su implementación.

Supongamos que deseamos desarrollar un controlador borroso utilizando la implicación de Mamdani, y que la base de reglas contiene una única regla de inferencia:

$$\text{Regla : SI } x \text{ es } A \text{ ENTONCES } y \text{ es } B \quad (14.10)$$

y donde las reglas de pertenencia de los conjuntos borrosos A y B se definen como en la Figura 14.14.

Nombre	Operador de implicación
Zadeh Max-Min	$(\mu_A(x) \wedge \mu_B(y)) \vee (1 - \mu_A(x))$
Mamdani (min)	$\mu_A(x) \wedge \mu_B(y)$
Larsen (prod)	$\mu_A(x) \cdot \mu_B(y)$
Lukasiewicz (aritmético)	$1 \wedge (1 - \mu_A(x)) + \mu_B(y)$
Kleene-Dienes (booleano)	$(1 - \mu_A(x)) \vee \mu_B(y)$
Producto Acotado	$0 \vee (\mu_A(x) + \mu_B(y) - 1)$
Producto Drástico	$\begin{array}{ll} \mu_A(x) & \text{if } \mu_B(y) = 1 \\ \mu_B(x) & \text{if } \mu_A(y) = 1 \\ 0 & \text{if } \mu_A(y) < 1, \mu_B(y) < 1 \end{array}$

Tabla 14.1: Listado de operadores de implicación más utilizados en control.

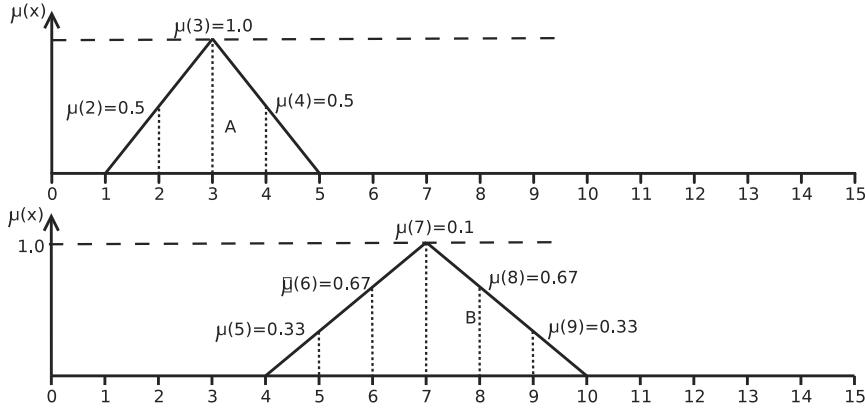
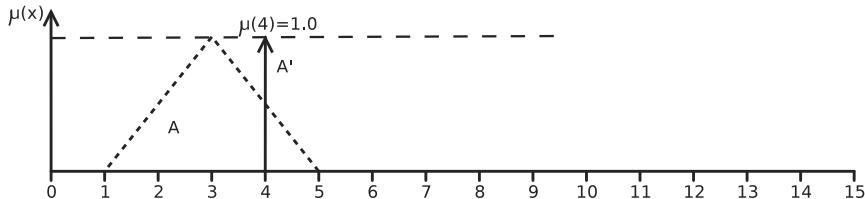


Figura 14.14: Ejemplo de descripción de variables LHS/RHS.

Supongamos que el controlador recibe, en un instante dado, una entrada x que vale exactamente 4. Mediante el proceso de fuzzificación, este valor se puede representar mediante un conjunto borroso A' definido por una función de pertenencia bivaluada $\mu_{A'}$ (fuzzificador unitario). Como puede observarse, el conjunto borroso A' no concuerda exactamente con A . ¿Qué podemos inferir sobre el valor de la variable y ?

Figura 14.15: Ejemplo de conjunto borroso A y fuzzificación unitaria (A') de x .

x_i	y_i	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0.33	0.5	0.5	0.5	0.33	0	0
3	0	0	0	0	0	0.33	0.67	1.0	0.67	0.33	0	0
4	0	0	0	0	0	0.33	0.5	0.5	0.5	0.33	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 14.2: Ejemplo de definición de la relación $R_{A \rightarrow B}$.

Representando los conjuntos A, B, y A' de forma algebraica (véanse las Figuras 14.15 y 14.14) tenemos que:

$$A = \sum \mu_A(x)/x = 0,5 / 2 + 1,0 / 3 + 0,5 / 4 \quad (14.11)$$

$$B = \sum \mu_B(y)/y = 0,33 / 5 + 0,67 / 6 + 1,0 / 7 + 0,67 / 8 + 0,33 / 9 \quad (14.12)$$

$$A' = \sum \mu_{A'}(x)/x = 0 / 2 + 0 / 3 + 1,0 / 4 \quad (14.13)$$

donde ambas variables, x e y están definidas sobre el mismo universo: $X = Y = \{0, 1, 2, \dots, 10\}$. Calculamos la función de pertenencia de la implicación usando el operador min de Mamdani:

$$\mu_{A \rightarrow B}(x_i, y_i) = \mu_A(x_i) \wedge \mu_B(y_i) \quad (14.14)$$

Por ejemplo:

$$\mu_R(2, 3) = \mu_A(2) \wedge \mu_B(3) = \min\{0,5, 0,0\} = 0 \quad (14.15)$$

$$\mu_R(2, 5) = \mu_A(2) \wedge \mu_B(5) = \min\{0,5, 0,33\} = 0,33 \quad (14.16)$$

$$(14.17)$$

Así, realizando un análisis para todos los posibles valores del universo, la relación $R_{A \rightarrow B}$ estaría definida por la Tabla 14.2.

14.4.5.2 Reglas de inferencia

Las inferencias borrosas son procedimientos computacionales para evaluar descripciones lingüísticas. Así en el capítulo 2 se describió cómo el Modus Ponens Generalizado o GMP (del inglés, Generalized Modus Ponens), que representa la forma habitual de inferencia a partir de reglas en los sistemas de control (véase la Figura 14.16). El GMP permite inferir el valor borroso B' , dado un valor de entrada A' y una relación de implicación $R_{A \rightarrow B}(x, y)$ que relacione ambas variables.

Premisa 1: SI x es A ENTONCES y es B
Premisa 2: x es A'
Consecuente: y es B'

Figura 14.16: Modus Ponens General (GMP).

El valor inferido B' se calcula mediante la composición del valor A' con la relación de implicación $R_{A \rightarrow B}(x, y)$. Por tanto:

$$B' = A' \circ R_{A \rightarrow B}(x, y) \quad (14.18)$$

$$\mu_{B'(y)} = \bigvee_{\forall x \in X} [\mu_{A'}(x) \wedge \mu_{A \rightarrow B}(x, y)] \quad (14.19)$$

Para el ejemplo que estamos considerando y un valor concreto $y = 5$, podemos obtener $\mu_{B'(5)}$ como:

$$\mu_{B'(5)} = \bigvee_{\forall x \in X} [0 \wedge 0,33, 0 \wedge 0,33, 1 \wedge 0,33] = \bigvee_{\forall x \in X} [0, 0, 0,33] = 0,33 \quad (14.20)$$

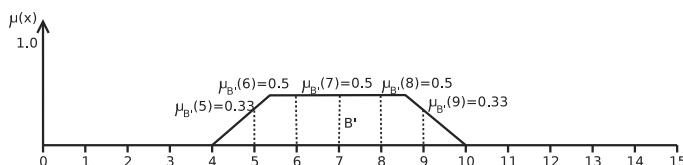
En forma matricial, podemos obtener B' como:

$$B'(y_j) = A'(x_i) \circ R_{A \rightarrow B}(x_i, y, i) = [0 \ 0 \ 1] \circ \begin{bmatrix} 0,33 & 0,50 & 0,50 & 0,50 & 0,33 \\ 0,33 & 0,67 & 1,00 & 0,67 & 0,33 \\ 0,33 & 0,50 & 0,50 & 0,50 & 0,33 \end{bmatrix} \quad (14.21)$$

resultando un conjunto B' dado por:

$$B' = 0,33 / 5 + 0,50 / 6 + 0,50 / 8 + 0,33 / 9 \quad (14.22)$$

que gráficamente se representa como en la Figura 14.17.

Figura 14.17: Conjunto borroso B' resultante de una inferencia.

Como puede observarse, el resultado B' es la función de pertenencia de B , recorrida a una altura igual al grado en el que A' concuerda con A (véase la Figura 14.18). Al grado de concordancia entre A y A' se le denomina grado de satisfacción de la

Operador Implicación	Interpretación agregación
Zadeh Max-Min	AND(\wedge)
Mamdani (min)	OR(\vee)
Larsen (prod)	OR(\vee)
Aritmético	AND(\wedge)
Booleano	AND(\wedge)
Producto Acotado	OR(\vee)
Producto Drástico	OR(\vee)

Tabla 14.3: Relación entre implicación y agregación.

regla o DOF (del inglés Degree Of Fulfillment). En el ejemplo anterior el grado de satisfacción es $DOF(A', R_{A \rightarrow B}) = 0,5$. De forma general, se calcula como:

$$DOF(A', R_{A \rightarrow B}) = \bigvee_{\forall x \in X} [\mu_A(x) \wedge \mu_{A'}(x)] \quad (14.23)$$

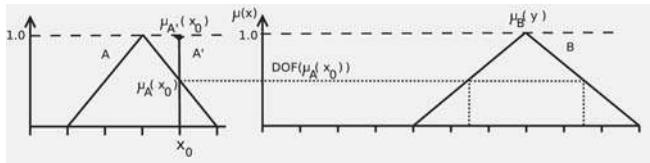


Figura 14.18: Efecto de recorte del Grado de Satisfacción (DOF).

Sin embargo, este efecto de recorte es una característica asociada al uso del operador \min de Mamdani, y no se produce con otros operadores de implicación. Así, por ejemplo, para el producto de Larsen, B' tiene la misma forma que B , pero escalada a una altura dada por el DOF .

14.4.5.3 Agregación para componer una solución

Cuando se activa más de una regla, es necesario componer los resultados de las distintas reglas en una única solución. Las reglas activadas son aquellas para las que $DOF \neq 0$. Para obtener una única salida, el controlador borroso debe *fusionar* los conjuntos borrosos resultantes de las inferencias en un único conjunto. Por ejemplo, si se ha utilizado el operador de Mamdani, hay que agregar los conjuntos de los consecuentes de las reglas activadas, pero recortados a nivel DOF .

La operación de agregación de conjuntos borrosos puede interpretarse como una conjunción de las soluciones parciales (AND) o una disyunción de las mismas (OR). Sin embargo, el operador de implicación seleccionado (Zadeh, Mamdani, Larsen ...etc.) condiciona la forma en la que se compone el resultado de las reglas activadas, es decir, el tipo de operador de agregación. En la Tabla 14.3 puede verse una colección de operadores de implicación y su operador de agregación correspondiente.

Como ejemplo, considérese la base de reglas de la Figura 14.13, donde los posibles valores de las variables borrosas de los antecedentes vienen definidos por las funciones de pertenencia de la Figura 14.12. Supóngase que, en $t = 0$, las variables de entrada toman los valores precisos $x_0 = 4$ y $z_0 = 5$. Se activarán las reglas R1, R2 y R3 de la base de reglas (véase la Figura 14.19).

R1: SI x es A ENTONCES y es B
R2: SI z es C ENTONCES y es D
R3: SI x es A Y z es C ENTONCES y es F

Figura 14.19: Base de reglas borrosas.

En primer lugar, se realizará la fuzzificación unitaria de las variables de entrada x y z del siguiente modo $\mu_{A'}(x_0) = 1$ y $\mu_{C'}(z_0) = 1$. Utilizando el operador de Mamdani, se calcula el DOF para las reglas activadas:

$$DOF_1 = \mu_A(x_0) = 0,5 \quad (14.24)$$

$$DOF_2 = \mu_C(z_0) = 0,66 \quad (14.25)$$

$$DOF_3 = \mu_A(x_0) \wedge \mu_C(z_0) = 0,5 \quad (14.26)$$

$$(14.27)$$

Así, la regla R_1 contribuye a la solución con $\mu_{B'}(y)$, que es el conjunto definido por $\mu_B(y)$ recortado a la altura DOF_1 . Del mismo modo R_2 contribuye a la solución con $\mu_{D'}(y)$ ($\mu_D(y)$ recortado por DOF_2) y R_3 contribuye a la solución con $\mu_{F'}(y)$ ($\mu_F(y)$ recortado a la altura DOF_3). En la Figura 14.20 podemos ver un resumen del proceso.

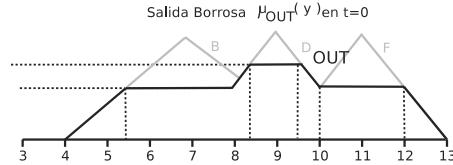
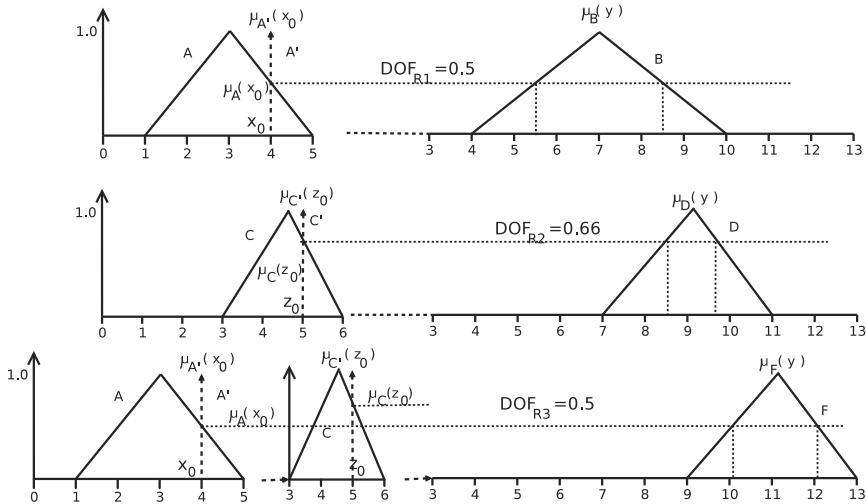
Finalmente, la salida global se calcula utilizando el operador de agregación OR (véase la Tabla 14.3). En la Figura 14.21 puede observarse la salida global producida por el mecanismo de inferencias borrosas, que es:

$$\mu_{OUT}(y) = \mu_{B'}(y) \vee \mu_{D'}(y) \vee \mu_{F'}(y) \quad (14.28)$$

14.4.6 Obtención de salidas precisas: defuzzificación

Una vez que la entrada ha sido procesada por el mecanismo de inferencias borrosas, se obtiene una salida global, dada por un conjunto borroso $\mu_{OUT}(y)$. En un controlador borroso, la defuzzificación se utiliza para elegir un valor de salida preciso y^* , a partir de $\mu_{OUT}(y)$. La defuzzificación es el proceso consistente en obtener un valor preciso representativo de un conjunto borroso.

Se han propuesto numerosas técnicas de defuzzificación a lo largo de los años. No existe una técnica general que supere a las demás, sea cual sea la aplicación. Por tanto, la elección de un método concreto tiene un impacto muy importante sobre la eficiencia y precisión del controlador.



Las técnicas de defuzzificación más utilizadas son:

- Centroide o Centro de las Áreas (COA, del inglés Center of Areas).
- Centro de las Sumas (COS, del inglés Center of Sums).
- Media de los Máximos (MOM, del inglés Mean of Maxima).
- Promedio de los Centros (CA, del inglés Center Average).

El Centroide (COA) calcula y^* como el centro geométrico del valor borroso de salida $\mu_{OUT}(y)$. El centro geométrico y^* de $\mu_{OUT}(y)$ es el punto que divide el área bajo la curva $\mu_{OUT}(y)$ en dos regiones de igual área. En un dominio discreto se define como:

$$y^* = \frac{\sum_{i=1}^N y_i \mu_{OUT}(y_i)}{\sum_{i=1}^N \mu_{OUT}(y_i)} \quad (14.29)$$

donde la suma se extiende a los N valores y_i del dominio discreto. En el caso de un dominio continuo, el sumatorio se convierte en una integral.

El COA tiende a favorecer los valores centrales sobre los periféricos y sólo cuenta una vez las áreas solapadas, puesto que opera sobre la función $\mu_{OUT}(u)$ obtenida tras la agregación. El Centro de las Sumas (COS) resuelve este último problema, ya que opera separadamente sobre cada contribución. Para un dominio discreto, el COS se define como:

$$y^* = \frac{\sum_{i=1}^N y_i \sum_{k=1}^M \mu_{G'_k}(y_i)}{\sum_{i=1}^N \sum_{k=1}^M \mu_{G'_k}(y_i)} \quad (14.30)$$

donde $\mu_{G'_k}(y)$ es la función de pertenencia resultante de aplicar la regla k-ésima.

La Media de los Máximos (MOM) calcula y^* como el valor medio de los puntos de máximo grado de pertenencia en la salida global $\mu_{OUT}(y)$. Esta técnica es muy rápida y no muestra preferencias por los valores centrales, como sí ocurre con el COA y el COS. Sin embargo, no tiene en cuenta la forma de $\mu_{OUT}(y)$. El MOM se calcula como:

$$y^* = \sum_{j=1}^P \frac{u_j}{P} \quad (14.31)$$

donde u_j es el j -ésimo máximo de $\mu_{OUT}(y)$ y P es el número total de máximos.

El Promedio de los Centros (CA) opera separadamente sobre cada contribución y calcula un promedio ponderado de los centros de las contribuciones. Se define como:

$$y^* = \frac{\sum_{k=1}^M \bar{y}_k \mu_{G'_k}(\bar{y}_k)}{\sum_{k=1}^M \mu_{G'_k}(\bar{y}_k)} \quad (14.32)$$

donde $\mu_{G'_k}(y)$ es la función de pertenencia resultante de aplicar la regla k-ésima e \bar{y}_k es el centro del conjunto borroso G'_k .

14.5 Arquitecturas y herramientas

14.5.1 Herramientas para control borroso

En el desarrollo de sistemas de control borroso resulta muy útil el uso de herramientas que ayuden al diseño y a la implementación. Existe un amplio abanico de herramientas para el desarrollo de sistemas basados en lógica borrosa, tanto comerciales como gratuitas. Existen herramientas de carácter general, otras especializadas en dominios concretos de la industria y otras que asisten en etapas concretas del desarrollo de sistemas borrosos.

Las herramientas de carácter general (*Fuzzy Logic Toolbox/MATLAB* o *Fuzzy-TECH*) son las más extendidas, tanto en el mundo académico como el profesional. Sin embargo, existen otras muchas herramientas útiles a la hora de desarrollar estos sistemas. *FuzzyCLIPS*¹ (de Togai Infracologic) es un paquete software que proporciona métodos para el diseño, la depuración y el test de sistemas expertos basados en lógica borrosa. *FIDE*² (de Apronix) es un conjunto de herramientas para facilitar el desarrollo de productos basados en lógica borrosa integrado con microcontroladores comerciales, ofreciendo ayudas para la depuración, la simulación y control en tiempo real.

Si se desea facilitar la programación de sistemas basados en lógica borrosa, existen diferentes librerías que simplifican esta tarea. Por ejemplo, *mbFuzzIT*³ es una librería gratuita para programación en Java, que incluye un editor gráfico como ayuda al desarrollo. *jFuzzyLogic*⁴ es otro paquete basado en Java, que permite la selección de distintas funciones de pertenencia, métodos de defuzzificación y métodos de implicación. Una alternativa más completa es la utilización de entornos de desarrollo para la programación de sistemas guiada por un framework general, como es el caso del framework *xFuzzy*⁵.

En el resto de la sección, se describen las características principales de algunas de las herramientas anteriormente mencionadas.

FuzzyTECH

*FuzzyTECH*⁶ es un producto propiedad de *INFORM GmbH and Inform Software Corp.* especializado en el desarrollo de aplicaciones técnicas que utilizan lógica borrosa.

Esta herramienta emplea una representación de números borrosos mediante funciones de pertenencia trapezoidales. Una de las características, a la que *FuzzyTECH* presta especial atención es la fuzzificación. La herramienta ofrece diferentes técnicas para el cálculo eficiente de la fuzzificación, e incluye algunas orientadas a la implementación hardware. *FuzzyTECH* permite usar algunos de los métodos de defuzzificación más habituales (CA, COA y MOM), así como otros utilizados en el reconocimiento de patrones (HyperCoM).

Una de las características fundamentales de *FuzzyTECH* es la generación automática de código C. Cuando éste es compilado se obtienen resultados muy eficientes, lo cual es una característica muy atractiva para el desarrollo de aplicaciones profesionales. Además de versiones simplificadas con objetivos académicos, *FuzzyTECH* también ofrece diferentes extensiones para sistemas de control embebidos y desarrollos especiales para la industria de la automoción.

¹<http://www.ortech-engr.com/fuzzy/fzyclips.html>

²<http://www.aptronix.com/fide/fide.htm>

³http://mbfuzzit.sourceforge.net/en/mbfuzzit_software.html

⁴<http://jfuzzylogic.sourceforge.net/html/index.html>

⁵<http://www.imse.cnm.es/Xfuzzy>

⁶<http://www.fuzzytech.com>

Fuzzy Logic Toolbox/Simulink/MatLab:

*Fuzzy Logic Toolbox*⁷(FLT) es un producto de *The Mathworks*⁸ para el entorno *MATLAB*⁹. La *FLT* es una ampliación del entorno de desarrollo *MATLAB* para el diseño de sistemas basados en lógica borrosa. Ofrece una entorno gráfico muy sencillo para el desarrollo de este tipo de sistemas paso a paso. Esta toolbox facilita el diseño de las reglas y la selección de las funciones más utilizadas en lógica borrosa. Las principales características de *FLT* son: Interfaces gráficas sencillas para la definición de variables LHS/RHS, edición de funciones de pertenencia, descripción de reglas y visualización de las reglas activadas.

Además, la *FLT* puede utilizarse de manera independiente o junto al entorno de desarrollo *Simulink* (también para *MATLAB*), orientado a la simulación de sistemas dinámicos. En la sección 14.5.2 puede verse un ejemplo de la realización de un sistema de control borroso completo con esta herramienta.

Xfuzzy

El entorno *Xfuzzy*¹⁰ es un desarrollo del Centro Nacional de Microelectrónica del CSIC (Centro Superior de Investigaciones Científicas) en España. *Xfuzzy* es un entorno de desarrollo para sistemas de inferencia basados en lógica borrosa. Está formado por un conjunto de herramientas que facilitan las distintas etapas del proceso de diseño, desde su descripción inicial hasta la implementación final. En la actualidad se encuentra disponible la versión *Xfuzzy 3.0*, desarrollada en Java.

Xfuzzy propone una metodología de desarrollo de sistemas borrosos basada en cuatro etapas: descripción, verificación, ajuste y síntesis. La etapa de descripción ofrece herramientas visuales para la definición del sistema borroso. La etapa de verificación permite la simulación del sistema. La etapa de ajuste permite la utilización de algoritmos de aprendizaje para configurar variables. Por último, la etapa de síntesis incluye herramientas para generar descripciones en lenguajes de alto nivel para implementaciones software o hardware. *Xfuzzy* está basado en el lenguaje *XFL3*, que permite definir bases de reglas jerárquicas, modificadores lingüísticos, funciones de pertenencia y métodos de defuzzificación. Una de las características más destacables es el amplio abanico de técnicas y métodos disponibles en este lenguaje.

14.5.2 Ejemplo de desarrollo de un controlador borroso

El tanque de agua

Se desea diseñar un controlador borroso para mantener constante el nivel de agua de un tanque (véase la Figura 14.22). Para ello, se puede ajustar la cantidad de agua que entra en el contenedor mediante una válvula. En la base del tanque existe una tubería que drena, de forma continua, el agua contenida. En [Hines, 1997] se propone

⁷<http://www.mathworks.com/products/fuzzylogic/>

⁸<http://www.mathworks.com>

⁹<http://www.mathworks.com/products/matlab/>

¹⁰<http://www.imse.cnm.es/Xfuzzy>

un conjunto sencillo de reglas que permite controlar, de forma bastante eficaz, el nivel de líquido en un problema similar. En esta sección, se extiende dicho ejemplo y se muestra su implementación utilizando *Simulink* de *MATLAB*.

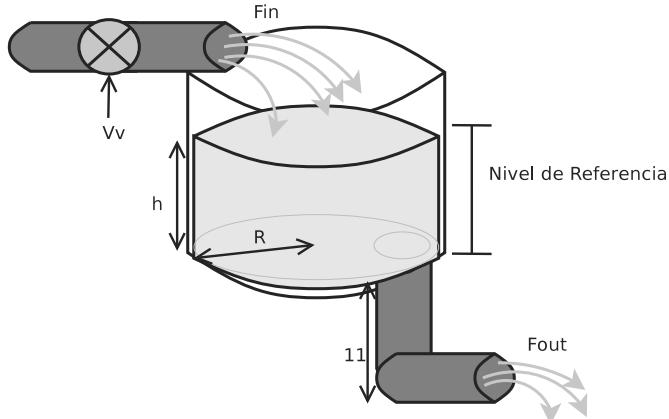


Figura 14.22: Ejemplo del tanque de agua.

Identificación de elementos principales

En primer lugar hay que identificar las variables del sistema. La variable de salida controlada es el nivel de líquido en el tanque. La entrada de referencia es el nivel de agua que se desea mantener. La señal de error es la diferencia entre el nivel actual del agua del contenedor y el nivel de referencia elegido. La señal de control regula la válvula de salida del agua para abrirla o cerrarla, de forma gradual, y modificar así el nivel de agua del contenedor. La planta está formada por el contenedor de agua, la tubería drenante y la válvula de entrada de agua.

Modelado de la planta

Antes de diseñar el controlador borroso, es necesario modelar el comportamiento de la planta. Dado que el tanque tiene forma cilíndrica, el volumen de agua V contenido por el tanque en un instante dado es igual al producto del área de la base del tanque A_c por la altura del líquido h :

$$V = A_c h = \pi R^2 h \quad (14.33)$$

donde R es el radio de la base del tanque.

Por otra parte, el volumen de agua varía a lo largo del tiempo, y sus variaciones dependen del flujo de entrada de agua F_{in} y del flujo de salida de agua F_{out} :

$$\dot{V} = F_{in} - F_{out} \quad (14.34)$$

donde \dot{V} representa la primera derivada del volumen respecto del tiempo. Por tanto, las variaciones del nivel de líquido vendrán dadas por:

$$\dot{h} = (F_{in} - F_{out})/A_c \quad (14.35)$$

Los flujos de entrada y salida pueden calcularse mediante las expresiones:

$$F_{out} = k_1 \sqrt{h + k_2} \quad (14.36)$$

$$F_{in} = k_3 V_c \quad (14.37)$$

donde k_1 es una constante que depende de las dimensiones del tanque, k_2 es la altura de la parte acodada del tubo de salida, V_c es una tensión eléctrica que regula la posición de la válvula de entrada, y k_3 es una constante que establece una relación proporcional entre la tensión de control aplicada a la válvula y el grado de apertura que provoca en la misma.

Finalmente, la ecuación diferencial que gobierna el comportamiento de la planta puede obtenerse a partir de las expresiones anteriores como:

$$A_c \dot{h} + k_1 \sqrt{h + k_2} = k_3 V_c \quad (14.38)$$

Obsérvese que la dinámica de la planta es no lineal, debido a que las leyes hidrodinámicas establecen que el flujo de salida es proporcional a la raíz cuadrada de la altura de la columna de líquido.

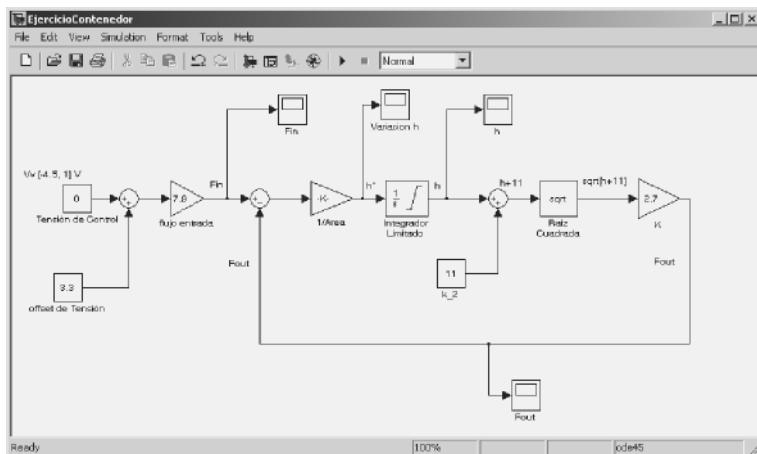


Figura 14.23: Planta a controlar realizada en Simulink de MATLAB.

El siguiente paso consiste en construir un modelo de la planta en *Simulink*, que permitirá simular dinámicamente su comportamiento. Para ello, se establecen las

siguientes constantes: $k_1 = 2,7$, $R = 3$, $k_2=11$ y $k_3 = 7,9$. Con estos valores, las relaciones que se deben implementar en Simulink son:

$$F_{in} = V_v * 7,9 \quad (14.39)$$

$$F_{out} = 2,7 * \sqrt{h + 11} \quad (14.40)$$

$$\dot{h} = (F_{in} - F_{out})/(\pi 9) \quad (14.41)$$

Adicionalmente, se supone que las especificaciones del fabricante de la válvula establecen que la tensión eléctrica aplicada a la válvula debe estar en el rango $V_c = [-1,2; 4,3]V$. Esta tensión se va a obtener sumando una tensión constante (offset) de $3,3V$ a una tensión variable V_v en el rango $[-4,5; 1]V$. Es decir, $V_c = V_v + 3,3V \in [-4,5; 1] + 3,3V = [-1,2; 4,3]V$. La variable V_v será la señal de control.

En la Figura 14.23 se muestra un modelo de la planta implementado en *Simulink*, que tiene en cuenta todas estas consideraciones.

Las gráficas de la Figura 14.24 muestran varias simulaciones del comportamiento de la planta durante 100 segundos, obtenidas aplicando distintas tensiones de control V_v . Cuando $V_v = 0$ puede observarse cómo el flujo de salida va aumentando hasta llegar a la saturación (desbordamiento del tanque) al llegar a los 90 segundos. Cuando la válvula está en apertura máxima, es decir $V_v = 1$, el tanque llega a la saturación mucho antes (a los 45 segundos aproximadamente). Cuando la válvula se encuentra bastante cerrada ($V_v = -1,5$) se consigue que el tanque no se desborde durante el tiempo de simulación.

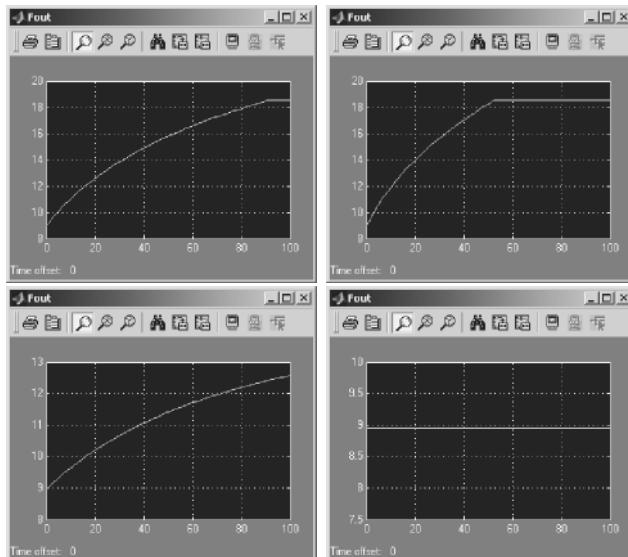


Figura 14.24: Salidas de la simulación del comportamiento de la planta con $V_v = 0$, $V_v = 1$, $V_v = -1,5$ y $V_v = -4,5$.

Para facilitar el modelado del sistema de control, todos los componentes del modelo de la planta se agruparán como un subsistema *Simulink* al que se denomina *Contenedor*.

Diseño del controlador borroso

Dado que la dinámica de la planta es no lineal, resulta apropiado utilizar técnicas de control borroso para desarrollar el sistema de control.

En primer lugar se establecerá cuáles son las entradas y salidas del controlador y se definirá la arquitectura general del sistema que va a controlar la planta. Como ya se comentó anteriormente, la señal de error que debe utilizar el controlador borroso es la diferencia entre el nivel de agua deseado y el nivel de agua del tanque en cada instante, es decir, el error producido. Sin embargo, en este problema también resulta conveniente considerar el grado de crecimiento o decrecimiento del nivel del agua en cada instante de tiempo. La velocidad con la que el nivel del agua sube o baja tiene un valor informativo muy importante para poder controlar mejor el tanque, ya que para velocidades muy grandes de crecida sería necesario que el controlador reaccionara con una apertura brusca de la válvula y análogamente para una bajada repentina. Por lo tanto, el controlador borroso tendrá dos entradas:

- (1) **error:** El error entre el nivel de agua esperado y el nivel del tanque y
- (2) **velocidad-h:** La velocidad de crecimiento del nivel del tanque, es decir, la derivada del nivel respecto al tiempo.

La salida del controlador (**V-válvula**) será la tensión V_v que controla la válvula de entrada del tanque de agua. Así, el sistema de control de la planta ofrece una topología de controlador de lazo cerrado (véase la Figura 14.25).

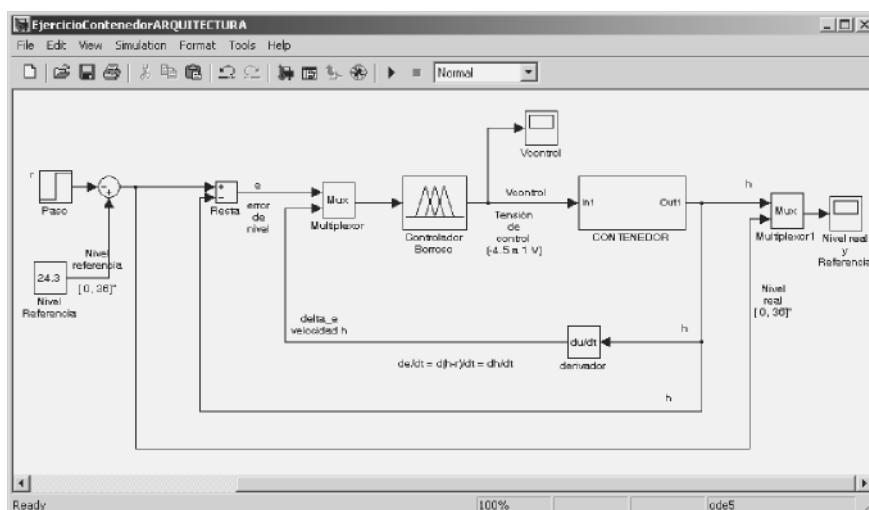


Figura 14.25: Arquitectura del sistema de control borroso.

En segundo lugar se diseñará un controlador borroso, dadas las entradas y salidas descritas. Para ello se utilizará la *Fuzzy Logic Toolbox* que ofrece *MATLAB*. Con la ayuda de un experto se definirán los valores de las variables LHS / RHS que se usarán en las reglas borrosas del controlador. Por tanto, hay que establecer conjuntos borrosos (etiquetas lingüísticas), tanto para las señales de entrada del controlador (la diferencia de nivel y la velocidad de cambio del nivel), como para la de salida (tensión de control de la válvula). El rango de variación de la señal de diferencia de nivel se establece en [-36,36], y se eligen las siguientes funciones de pertenencia para los conjuntos borrosos que definen sus valores:

$$\begin{aligned} \text{negativo grande : } & ng[-36, -36, -10, -5] (\text{trapezoidal}) \\ \text{negativo peque. : } & np[-10, -2, 0] (\text{triangular}) \\ \text{cero : } & c[-1, 0, 1] (\text{triangular}) \\ \text{positivo peque. : } & pp[0, 2, 10] (\text{triangular}) \\ \text{positivo grande : } & pg[5, 10, 36, 36] (\text{trapezoidal}) \end{aligned}$$

La señal de velocidad de cambio del nivel de agua (velocidad-h) se establece en el dominio [-40,40] y se eligen las siguientes etiquetas lingüísticas:

$$\begin{aligned} \text{positivo : } & p[0, 2, 40, 40] \\ \text{cero : } & c[-1, 0, 1] \\ \text{negativo : } & n[-40, -40, -2] \end{aligned}$$

En cuanto a la señal de salida (tensión de control de la válvula V_v), se define en el rango [-4.5,1], estableciendo las siguientes etiquetas lingüísticas:

$$\begin{aligned} \text{muy baja : } & mb[-4.5, -4.5, -4] \\ \text{baja : } & b[-3, -4, -3] \\ \text{media : } & m[-3, -2, -1] \\ \text{alta : } & a[-1, 0, 1] \\ \text{muy alta : } & ma[0, 1, 1] \end{aligned}$$

Estos valores se definen mediante los editores de la herramienta de implementación (véase las Figuras 14.26 y 14.27), seleccionando en cada caso el tipo de función de pertenencia (triangular o trapezoidal), así como los parámetros generales del controlador (tanto operadores de implicación y agregación, como métodos de fuzzificación y defuzzificación).

Uno de los factores fundamentales para el buen funcionamiento del controlador borroso es la correcta especificación de las reglas borrosas a utilizar. Así, es necesario que el experto del dominio establezca las reglas de forma apropiada. En este ejercicio se implementará el conjunto de reglas definido por [Hines, 1997] (Tabla 14.4), utilizando el editor de reglas de la *Fuzzy Logic Toolbox* de *MATLAB* (véase la Figura 14.28).

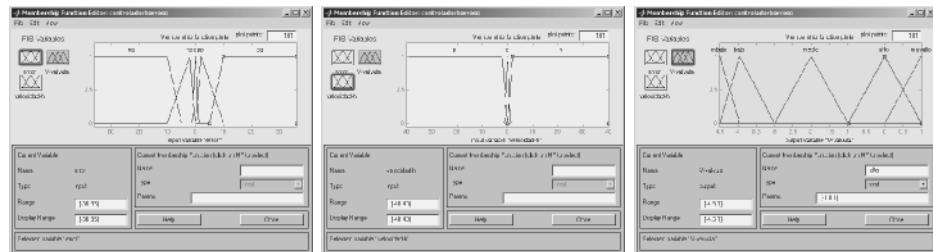


Figura 14.26: Edición de las funciones de pertenencia utilizando la *Fuzzy Logic Toolbox*.

IF error	AND velocidad-h	THEN V-válvula
ng	p	mb
ng	z	mb
ng	n	b
np	p	b
np	c	b
np	n	m
c	p	m
c	c	m
c	n	m
pp	p	m
pp	c	a
pp	n	a
pg	p	a
pg	c	ma
pg	n	ma

Tabla 14.4: Reglas del controlador borroso.

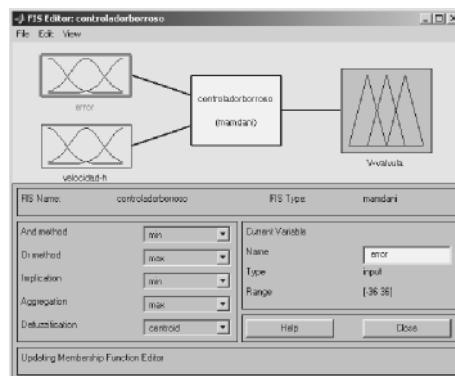


Figura 14.27: Ventana principal de la *Fuzzy Logic Toolbox* de MATLAB.

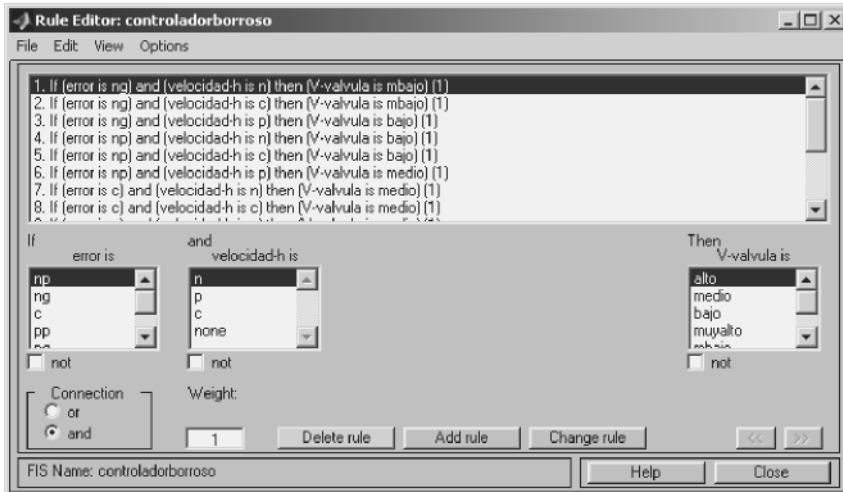


Figura 14.28: Edición de reglas borrosas utilizando *Fuzzy Logic Toolbox* de MATLAB.

Finalmente, se integra el controlador generado por la herramienta *Fuzzy Logic Toolbox* en *Simulink*, encapsulándolo en un bloque denominado **ControladorBorroso**, cuya salida se conectará directamente a la planta (bloque **Contenedor**), como muestra la Figura 14.25.

Para la simulación, se establece inicialmente una señal de referencia (nivel de agua esperado) con un valor de 24.3, saltando después a un segundo nivel de referencia fijado en 14.3. En *Simulink*, esto se implementa utilizando un bloque de tipo Step (*Paso*), que mantendrá un valor de 0 hasta llegar al segundo 50, pasando a tener un valor de 10 que se restará a la constante del nivel de referencia (24.3). Los resultados de la simulación (véase la Figura 14.29 izquierda) muestran cómo se adapta el nivel de agua del tanque a los valores de referencia establecidos, gracias a la acción del controlador borroso sobre la válvula de entrada de agua. Como puede observarse, transcurre un cierto tiempo tras el cambio brusco del nivel de referencia hasta que el nivel real se aproxima al esperado. En cuanto a la señal de control, se mantiene prácticamente estable hasta llegar al instante 50, en el que el controlador aplica una señal de apertura máxima de la válvula. Tras unos pocos segundos, cuando el nivel real comienza a acercarse al nuevo nivel de referencia, el controlador cierra de nuevo la válvula gradualmente, y después la mantiene estable pero ligeramente más cerrada que al principio (véase la Figura 14.29 derecha).

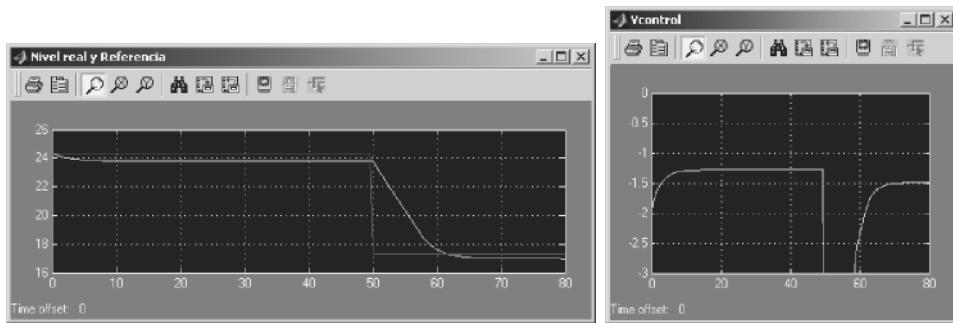


Figura 14.29: Durante la simulación: (izquierda) Niveles de agua esperado (rosa) y actual (amarillo).Valores de la señal de control (derecha).

14.6 Arquitecturas de pizarra

Las arquitecturas de pizarra son arquitecturas genéricas, es decir independientes de la tarea y de la aplicación. Prescriben la integración de múltiples módulos de resolución de problemas, que se comunican entre sí a través de un repositorio de datos globalmente accesible [Hunt, 2002]. Los módulos de resolución de problemas se denominan *fuentes de conocimiento* y la base de datos central se denomina *pizarra*. En esencia, se trata de una arquitectura para la resolución distribuida de problemas complejos.

Una de las principales características de este tipo de arquitectura software es su capacidad para soportar esquemas de razonamiento reactivo que permiten responder rápidamente a eventos ocurridos en el mundo exterior. Esto las hace idóneas para tareas en las que es necesario realizar interpretación de señales, y especialmente en aplicaciones al control y supervisión de procesos.

14.6.1 Metáfora de la pizarra

Las arquitecturas de pizarra se basan en una idea propuesta originalmente por Newell, conocida como *metáfora de la pizarra* [Newell, 1962], que posteriormente fue aplicada con éxito en sistemas como HEARSAY-II [Erman y otros, 1980]. Han desempeñado un papel relevante en la historia de la IA y han servido de fuente de inspiración para otras arquitecturas más novedosas, que han recogido y actualizado sus ideas básicas. En la actualidad, existen otras arquitecturas para la resolución distribuida de problemas que gozan de mayor popularidad, como es el caso de los sistemas multi-agente. No obstante, existen diferencias esenciales entre ambas aproximaciones, que delimitan el ámbito de aplicación preferencial de cada tipo de arquitectura.

La idea original de Newell consistía en un modelo idealizado de la resolución de problemas en grupo, y se puede formalizar en los siguientes términos [Corkill, 1991; Newell, 1962]:

- Imaginemos un grupo de expertos humanos que deben trabajar cooperativamente para resolver un problema complejo. Cada experto puede ser especialista en un aspecto distinto del problema.
- Los expertos están ubicados en una sala de reuniones, donde disponen de una pizarra que usarán como espacio de trabajo para desarrollar la solución. La pizarra será su único medio de comunicación.
- Una vez escritos los datos iniciales del problema en la pizarra, los expertos pueden escribir, modificar o borrar elementos en la pizarra, así como trazar relaciones entre los elementos.
- Los expertos deben examinar la información contenida en la pizarra, y sólo pueden alterarla cuando alcanzan alguna conclusión que pueda contribuir a la solución y que corresponda a su área de experiencia. Al escribir nueva información en la pizarra, darán una oportunidad a otros expertos para aplicar su experiencia.
- Este proceso continúa hasta que se alcanza una solución final completa.
- Existe un moderador que coordina los turnos para escribir en la pizarra, estableciendo unas prioridades en términos del estado del problema.

La metáfora de la pizarra de Newell, aunque simple, estableció las características principales de los *sistemas de pizarra*. Algunos años después, comenzaron a aparecer las primeras implementaciones basadas en estas ideas. Una aplicación pionera, que sentó las bases para desarrollos posteriores, fue el sistema HEARSAY-II [Erman y otros, 1980], orientado al reconocimiento de voz en tiempo real a partir de un vocabulario limitado. Una aportación original de HEARSAY-II fue la introducción de *niveles de abstracción* en la pizarra. En una organización de la pizarra por niveles de abstracción, el nivel inferior corresponde a los datos de entrada y el nivel superior contendrá la solución final. Los especialistas avanzan por los niveles intermedios, llenándolos con información que eventualmente permitirá construir la solución final. En HEARSAY-II había niveles correspondientes a señal acústica, propiedades espectrales, fonemas, sílabas, morfemas, palabras, frases y sentencias.

14.6.2 Componentes básicos de un sistema de pizarra

Existen múltiples variantes de las arquitecturas de pizarra, por lo que es habitual tomar como punto de referencia una de las propuestas más simples: la correspondiente al sistema HEARSAY-II, a la que se suele denominar *arquitectura básica de pizarra*. Consta de tres componentes (véase la Figura 14.30):

1. Una *Pizarra*: Es una base de datos central que contiene los datos de entrada y los elementos de una solución al problema construida progresivamente.
2. Un conjunto de *Fuentes de Conocimiento*: Módulos que contienen conocimiento sobre distintos aspectos del problema del dominio. Este conocimiento puede ser

aplicado a los hechos almacenados en la pizarra, produciendo nuevos datos que quedarán también registrados en la pizarra. El término fuente de conocimiento suele abreviarse como *KS* (*Knowledge Source* en inglés).

3. Un *Mecanismo de Control*: Es un único módulo central que examina el estado de la solución en la pizarra e inicia la ejecución de las KS más apropiadas para avanzar en la resolución del problema. El mecanismo de control tiene información sobre el tipo de conocimiento contenido en cada fuente. En función de ello, y de los cambios producidos en la pizarra, establece prioridades entre las KS y decide cual se aplicará a continuación. Implementa, por tanto, la estrategia de resolución del problema.

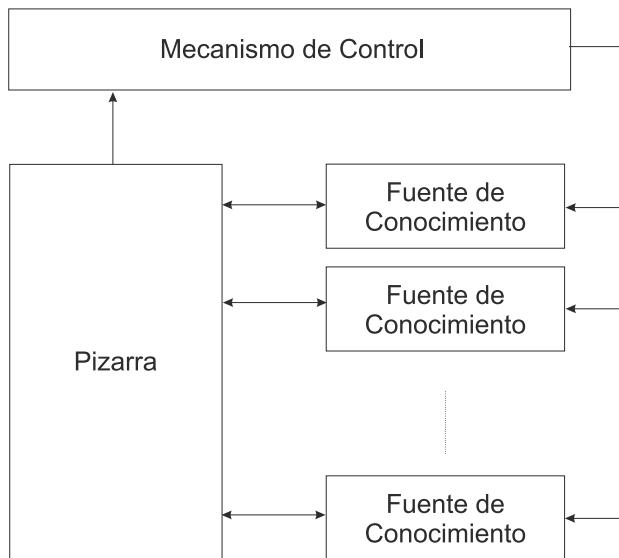


Figura 14.30: Componentes básicos de una arquitectura de pizarra.

14.6.3 Ciclo de control

Un principio básico de las arquitecturas de pizarra es que las KS son independientes y autoactivables. Son independientes porque su ejecución no depende explícitamente de la ejecución de otras KS y la única comunicación con otras KS es a través de la pizarra. Son autoactivables porque la activación de una KS se determina sólo a partir del estado actual de la pizarra.

En principio, por tanto, no haría falta un mecanismo de control central. Sin embargo, y en ausencia de la posibilidad de un intercambio de mensajes directo entre distintas KS, es preferible coordinar globalmente el orden de ejecución de las KS para garantizar que, en cada momento, se elijan las acciones más prometedoras. De esta forma, se evita perseguir líneas de razonamiento que tienen poca probabilidad de

contribuir a la solución. Además, se facilita el cambio dinámico del foco de razonamiento. Así, cuando llegan datos del mundo exterior que requieren mayor prioridad de atención, o que puedan explotarse para avanzar más rápidamente en la construcción de la solución, un mecanismo de control central puede reaccionar inmediatamente, cambiando el foco de razonamiento para sacar mayor partido de la nueva situación.

En las arquitecturas de pizarra se introduce un mecanismo de control central que aplica conocimiento para focalizar el proceso de razonamiento sobre el problema. El ciclo de control básico implementado por este mecanismo consta de tres fases:

1. Comparar: Determina las KS que deben ser disparadas y activadas.
2. Deliberar: Elige una KS para ser ejecutada, decidiendo cuál es la más apropiada para la situación actual.
3. Actuar: Se aplica la acción de la KS elegida.

La arquitectura básica de HEARSAY-II introdujo un mecanismo de control basado en *agenda*. Esto significa que utiliza una estructura de datos, la agenda, para almacenar las posibles actividades a realizar, y que en cada ciclo, las actividades son priorizadas, seleccionándose la de mayor prioridad.

Los componentes fundamentales de un mecanismo de control basado en agenda son (véase la Figura 14.31):

- Monitor de Pizarra: Identifica las KS que deben ser disparadas y activadas, a partir de los eventos producidos en la fase final del ciclo anterior. Si el evento es relevante para el conocimiento de una KS, se dice que la KS ha sido *disparada*. Si la KS es *aplicable*, es decir, si el estado global de la pizarra es tal que la KS puede desarrollar su función, se dice que la KS ha sido *activada*.
- Agenda: Almacena las KS activadas.
- Base de Datos del Foco del Control: Almacena información para determinar las "áreas más prometedoras" de la pizarra.
- Planificador: Establece unas prioridades entre las KS disponibles en la Agenda, en función del área de pizarra sobre la que es capaz de actuar cada KS activada y de la información suministrada por la base de datos del foco del control. Después, selecciona una KS de alta prioridad y ejecuta su parte de acción.

En la Figura 14.32 se muestra un esquema del ciclo de control básico. Cada recuadro representa el componente que se ejecuta en cada fase del ciclo. Las etiquetas de los arcos indican qué información se produce tras la aplicación de cada componente.

La fase inicial consiste en identificar las KS disparadas, y la implementa el monitor de pizarra. Cada KS tiene asociada una lista de los tipos de eventos en los que está interesada.

Cuando ocurre un cambio en la pizarra, éste se notifica al monitor de pizarra, que identifica el tipo de evento a que corresponde. El monitor de pizarra localiza las KS que deben ser disparadas a partir de las asociaciones entre KS y tipos de evento definidas al diseñar cada KS.

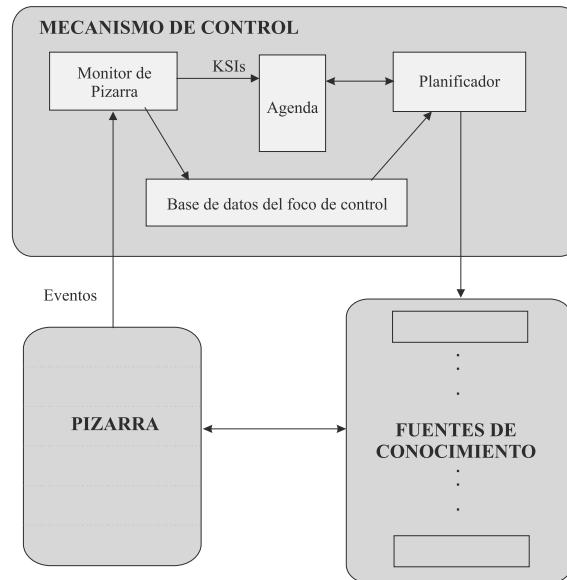


Figura 14.31: Componentes del mecanismo de control de una arquitectura básica de pizarra.

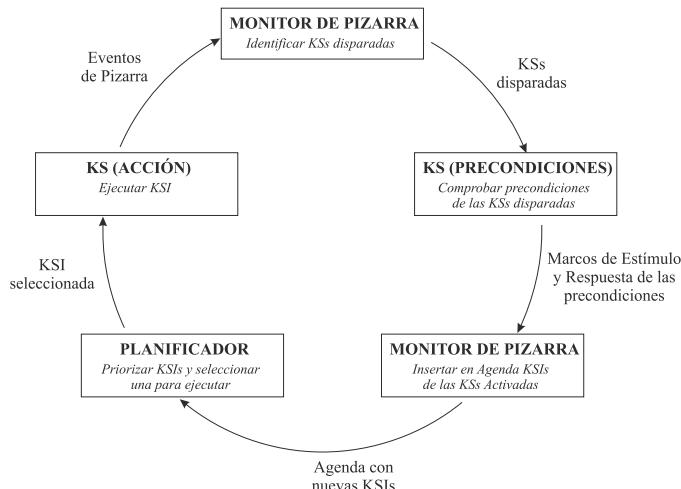


Figura 14.32: Fases del ciclo de control de una arquitectura básica de pizarra.

A continuación, el monitor de pizarra invoca las precondiciones de cada KS disparada, con el objetivo de determinar si debe quedar activada. La precondición de una KS es una pieza de código que retorna:

- Resultado: Verdadero o falso, según que la precondición se satisfaga o no.
- Información Contextual: Se usará en las siguientes fases del ciclo de control para elegir la KS a aplicar, y consiste en:
 - Marco de Estímulo: Estructura de datos que identifica los datos y/o las hipótesis que dieron lugar al disparo y activación de la KS.
 - Marco de Respuesta: Estructura de datos que identifica los efectos previsibles que se obtendrían si la KS fuese aplicada, es decir, los tipos de cambios que produciría en la pizarra y las áreas que serán afectadas.

Después, el monitor de pizarra retoma el control de la ejecución y decide si la KS queda activada, en función del resultado de la ejecución de su precondición. Para cada KS activada, se crea una *Instancia de Fuente de Conocimiento* (KSI o Knowledge Source Instance).

Cada KSI generada es almacenada en la Agenda. Una KSI es una estructura de datos que incluye toda la información necesaria para las siguientes fases (incluye los Marcos de Estímulo y Respuesta). Cada KSI representa una posible acción elegible por el sistema. En cada instante, las siguientes acciones que puede aplicar el sistema son únicamente las KSI contenidas en la Agenda.

El núcleo de la etapa deliberativa del ciclo de control corresponde a la ejecución del planificador, que constituye la siguiente fase del ciclo. El planificador establece la prioridad de cada KSI almacenada en la agenda. Las KSI quedarán ordenadas de mayor a menor prioridad. El planificador elige la KSI de mayor prioridad, la borra de la agenda y ejecuta la parte de acción de la correspondiente KS. Las restantes KSI permanecen en la agenda, por lo que continuarán siendo elegibles en siguientes ciclos de control, si alcanzan suficiente prioridad en las posteriores reevaluaciones de prioridades.

La prioridad de cada KSI se establece de forma absoluta, sin tener en cuenta las restantes KSI de la agenda. En la versión más simple, la prioridad se calcula aplicando una función lineal con pesos constantes a distintos factores.

La Base de Datos del Foco de Control contiene información sobre el estado global de la pizarra: las mejores hipótesis en cada área de pizarra y el tiempo transcurrido desde que se crearon. Por su parte, el Marco de Respuesta contiene información sobre las áreas de pizarra donde la KS crearía nuevas hipótesis, en caso de ser aplicada. Sabiendo cuáles son las mejores áreas de pizarra y cuáles son las áreas de pizarra sobre las que va actuar la KS, se pueden determinar los valores para los factores utilizados por el planificador para estimar la utilidad de las nuevas hipótesis.

La última fase del ciclo de control consiste en ejecutar la parte de acción de la KS elegida como prioritaria. Durante la ejecución, se producirán nuevos cambios en la pizarra, que darán lugar al inicio de un nuevo ciclo de control. En algunas implementaciones, se permite que la KS genere algunos comandos de control, como el comando :STOP, que daría lugar a la terminación del ciclo de control.

14.6.4 Características de las arquitecturas de pizarra

Las características principales de los *sistemas de pizarra* se pueden resumir en los siguientes puntos [Corkill, 1991]:

1. **Modularización y heterogeneidad de la experiencia:** Cada KS es especialista en resolver ciertas partes del problema, independientemente de las demás. Sólo precisa que la pizarra contenga la información estrictamente necesaria para resolver su parcela del problema, sin comunicación directa entre otros especialistas. Sus métodos de resolución de problemas pueden ser diferentes a los de otras KS. Esto facilita la implementación de sistemas híbridos que integren, por ejemplo, redes neuronales y sistemas basados en conocimiento.
2. **Lenguaje de interacción común:** La información contenida en la pizarra debe proporcionarse en un lenguaje de representación común, para que sea inteligible para las otras KS. El lenguaje debe ser lo suficientemente flexible para acomodar las contribuciones de los distintos especialistas.
3. **Niveles de abstracción:** En un problema complejo es importante subdividir la información, de forma que la pizarra quede organizada en regiones, cada una de ellas asociada a un tipo de información. De esta forma, a cada especialista le resulta más fácil localizar rápidamente la información relativa a su área de experiencia. Aunque la metáfora de Newell no lo planteaba directamente, las primeras implementaciones de sistemas de pizarra optaron por organizar la pizarra en niveles de abstracción.
Esta organización por niveles de abstracción puede convivir con otras organizaciones basadas en distintas dimensiones (tiempo, espacio, tipos enumerados, etc.)
4. **Construcción incremental de la solución:** Las soluciones completas son construidas pieza a pieza, y en diferentes niveles de abstracción, combinando gradualmente las contribuciones de los distintos especialistas. El sistema puede comutar dinámicamente entre distintos niveles de abstracción, persiguiendo líneas de razonamiento que pueden ser independientes, cooperativas o competitivas.
5. **Reactividad:** Se dice que un sistema implementa un *control reactivo* cuando es capaz de reaccionar a los cambios que ocurran en el entorno exterior en un plazo de tiempo aceptable. Las arquitecturas de pizarra son inherentemente reactivas: la entrada de nuevos datos externos en la pizarra provoca eventos que disparan y activan KS capaces de responder a ellos. Esto es eficiente porque, en lugar de tener que explorar toda la pizarra para localizar qué información nueva se ha introducido, es la pizarra la que informa de la ocurrencia de eventos de un tipo determinado y el sistema activa a los especialistas interesados en la ocurrencia de ese tipo de eventos. La respuesta a los cambios externos es rápida: en cada nuevo ciclo de control se determinan las nuevas KSI generadas por los nuevos eventos,

y entran a formar parte de las KSI elegibles. No hay que esperar a terminar un plan en curso para responder a la nueva información, como ocurriría en un sistema con razonamiento dirigido por objetivos.

6. **Oportunismo:** *Control oportunista* significa que el sistema es capaz de determinar y escoger como siguiente acción aquella que le permitirá avanzar mejor hacia la solución, dado el estado actual de la pizarra. El oportunismo implica la existencia de un mecanismo de refocalización: el sistema cambia el foco de atención para sacar partido de una nueva situación, entendiendo por nueva situación la disponibilidad de nuevos datos o de nuevas hipótesis más prometedoras que las previas. Las arquitecturas de pizarra son oportunistas porque en cada nuevo ciclo de control reevalúan todas las KSI disponibles en términos del estado actual (vuelven a calcular la prioridad, que puede haber cambiado al cambiar el estado de la pizarra). Para que un sistema sea oportunista debe ser forzosamente reactivo. Si el sistema no fuese reactivo a los cambios en el entorno, las KSI capaces de responder no serían identificadas inmediatamente, y el sistema no podría decidir refocalizar su atención hacia ellos. El inverso no es cierto. Un sistema reactivo no es necesariamente oportunista. Además de determinar las posibles respuestas a los cambios en el entorno, debe deliberar y escoger la mejor respuesta a la situación global.

En contrapartida, las arquitecturas de pizarra presentan algunos inconvenientes. Los dos principales son los siguientes:

1. El oportunismo introduce una sobrecarga en el control: En cada nuevo ciclo de control hay que volver a calcular las prioridades de las KSI antiguas que permanecen en la agenda. El oportunismo obliga a identificar la mejor KSI en función del nuevo estado de la pizarra. Además, en cada nuevo ciclo de control hay que determinar si las KSI antiguas, que permanecen en la agenda, son todavía aplicables. Los cambios en la pizarra pueden hacer que dejen de verificarse sus precondiciones.
2. El control oportunista se contrapone al control dirigido por objetivos. En el segundo caso, el sistema elige un objetivo, construye un plan para alcanzar el objetivo, y aplica secuencialmente los elementos del plan. Esto implica una descomposición de tareas en subtareas. Las arquitecturas de pizarra no aplican control dirigido por objetivos y su comportamiento puede llegar a parecer errático, si las KS no están bien diseñadas. Para algunas tareas, como por ejemplo el diagnóstico, el control dirigido por objetivos puede ser más apropiado que un control reactivo. Sin una gestión explícita de las metas el sistema no puede identificar correctamente las secuencias de acciones críticas para alcanzar los objetivos. Existen soluciones de compromiso, pero éste sigue siendo un problema abierto en sistemas para razonamiento: ¿Cómo conseguir arquitecturas capaces de operar de forma dirigida a objetivos sin sacrificar el oportunismo?

14.7 Sistemas en tiempo real basados en conocimiento

14.7.1 Sistemas en tiempo real

Antes de introducir el concepto de Sistema en Tiempo Real Basado en Conocimiento (STRBC), es necesario definir el concepto de Sistema en Tiempo Real (STR). Como se indica en Burns y Wellings [Burns y Wellings, 1997] existen diferentes interpretaciones sobre la naturaleza exacta del término Sistema en Tiempo Real; sin embargo, todas tienen en común la idea de respuesta en el tiempo, es decir, el tiempo que tarda el sistema en generar una salida a partir de las entradas correspondientes.

En el Oxford Dictionary of Computing se puede leer la siguiente definición de un STR: *Cualquier sistema en el que el tiempo en producir la respuesta es significativo. Esto es debido a que generalmente dichas respuestas se corresponden con algún cambio en el mundo real y las salidas que se producen están directamente relacionadas con dichos cambios. La diferencia de tiempo entre la salida y la entrada tiene que ser lo suficientemente pequeña para ser aceptable.* El término suficientemente pequeña depende del contexto. Para algunos sistemas puede ser del orden de milisegundos y para otros puede ser del orden de segundos.

En Randell et al. [Randell y otros, 1995] se puede leer la siguiente definición: "Un STR es un sistema al que se le requiere que reaccione a estímulos del entorno (incluido el paso del tiempo físico) dentro de intervalos de tiempos impuestos por dicho entorno".

Como se puede apreciar, estas definiciones cubren una gran variedad de sistemas. Si bien, a la mayor parte de los sistemas computacionales se les exige una respuesta en un tiempo finito, no se puede afirmar que el resultado sea erróneo si la salida no se produce dentro de un intervalo de tiempo prefijado. Esta característica puede ser considerada como la diferencia principal entre un STR y un sistema convencional. Por lo tanto, la corrección de un STR no sólo depende del resultado lógico la computación, sino también del tiempo que se tarda en producirlo.

Dependiendo de lo estricto que se sea en la aplicación de esta última afirmación, podemos distinguir entre Sistemas en Tiempo Real Duros (STRD) y Sistemas en Tiempo Real Suaves (STRS). Un STRD es un STR donde es absolutamente necesario que las respuestas ocurran antes del tiempo indicado (*tiempo límite*), mientras que en los STRS se permite que algunas respuestas puedan sobrepasar el tiempo límite. Por supuesto, en un mismo sistema podrán coexistir subsistemas de ambos tipos.

Como se puede apreciar, a partir de las definiciones anteriores, las características principales de un STR son:

- La existencia de limitaciones temporales en el tiempo de respuesta.
- La necesidad de interaccionar con el mundo real.

14.7.2 Concepto de sistema en tiempo real basado en conocimiento

Los STR son parte integral de muchas aplicaciones existentes hoy en día. Estas aplicaciones van desde los pequeños y sencillos controladores utilizados en gran parte de los electrodomésticos más comunes, hasta sistemas grandes y complejos utilizados en la industria y en entornos militares.

El aumento en la complejidad de las aplicaciones ha propiciado que aparezca un creciente interés en el uso de técnicas basadas en conocimiento en aplicaciones de tiempo real. Ello permite al ordenador emplear heurísticas y métodos para resolver problemas complejos, o definidos pobremente, que pueden ser impracticables con las técnicas convencionales. De esta forma, se pueden encontrar sistemas expertos aplicados a problemas en tiempo real que van desde los sistemas de bajo nivel que se utilizan para problemas de control hasta los de más alto nivel, como pueden ser la gestión de alarmas, predicción y diagnóstico de fallos, y la supervisión y coordinación de grupos de controladores. A este nivel se pueden agregar funciones de apoyo a estos sistemas, de tal forma que aparte de indicar las situaciones de alerta, se pueden incluir módulos que aconsejen al operador el modo de tratar dichas alarmas.

Una razón adicional para la introducción de sistemas expertos en tiempo real es la necesidad de reducir la carga cognitiva de los usuarios en ciertos dominios, para posibilitar un incremento en la productividad y evitar la producción de fallos humanos. Dentro de estas situaciones se pueden destacar aquellas en la que los humanos no pueden manipular simultáneamente toda la información relevante para alcanzar soluciones óptimas o en las que no se puede dar una solución con la suficiente rapidez.

Un STRBC es un sistema basado en conocimiento que necesita responder a un entorno cambiante, en el cual puede aparecer un flujo asíncrono de eventos, y que necesita adaptarse a cambios en los requerimientos debidos a limitaciones temporales, limitaciones hardware y limitaciones de recursos de cualquier tipo. Por lo tanto, este tipo de sistemas requerirá una arquitectura software flexible que proporcione capacidades de razonamiento sobre datos que cambian con cierta rapidez. El sistema estará sujeto a limitaciones en el tiempo de respuesta, y debe proporcionar capacidades de razonamiento temporal, no monotonicidad, manejo de interrupciones y métodos para tratar con entradas con ruido.

14.7.3 Características de los STRBC

Tradicionalmente, las técnicas de resolución de problemas basadas en conocimiento han sido aplicadas a dominios en los que los datos son estáticos y donde no se requieren tiempos de respuestas fijos. Estas características pueden ser apreciadas en muchos sistemas convencionales para diagnosis, clasificación, planificación, etc... Hay que tener en cuenta que el uso de sistemas expertos en problemas de tiempo real es relativamente reciente, y que no se han definido de forma clara los requerimientos que hay que exigir a estos sistemas. En los siguientes apartados, se describirán algunas de las características propias de los STRBC, sobre cuya necesidad existe amplio consenso.

14.7.4 Eventos asíncronos

Un sistema experto que vaya a ser usado en un entorno de tiempo real debe ser capaz de responder a entradas asíncronas o periódicas. Además, debe posibilitar la interrupción de sus funciones para aceptar entradas de eventos no previstos. Algunos eventos pueden ser más importantes que otros, por lo que el sistema debe posponer la atención a los eventos menos importantes para atender otros de mayor prioridad. Como ya se ha indicado, la capacidad de reaccionar inmediatamente ante sucesos asíncronos se denomina razonamiento reactivo. Hay que tener en cuenta que este tipo de razonamiento implica la existencia de un mecanismo de control que atienda periódicamente la ocurrencia de eventos o que pueda aceptar interrupciones.

Una característica subyacente a la capacidad de proporcionar razonamiento reactivo es la necesidad de que el STRBC perciba los posibles cambios que se produzcan en el entorno. Esto se consigue mediante la utilización de sensores que continuamente aportan al sistema información sobre la situación del entorno, permitiéndole reaccionar ante cambios no deseados del mismo.

14.7.5 Razonamiento no monótono

En un STRBC los datos, que pueden provenir de los sensores externos o bien de modificaciones internas, no permanecen estáticos durante toda la ejecución del sistema. Por tanto, los datos no son perdurables y tienden a perder su validez con el paso del tiempo. Esto puede dar lugar a que las conclusiones hechas en un determinado momento dejen de ser válidas posteriormente, bien porque la situación haya cambiado, bien porque se dispone de nueva información que permite corregir la conclusión. Ello hace necesario emplear técnicas de razonamiento no monótono. Es decir, hay que aplicar algún método que permita la retractación de conclusiones, o algún mecanismo, basado en técnicas de mantenimiento de la verdad, que mantenga la coherencia de todas las conclusiones a medida que se van obteniendo nuevos datos.

14.7.6 Modo de operación continuado

La mayoría de STR operan de forma continua hasta que son detenidos por el operador o por algún evento externo catastrófico. Esto implica que el descubrimiento de un fallo total o parcial en alguno de los componentes del sistema no tiene que suponer la parada del mismo. En aplicaciones de monitorización y control, el sistema tiene que estar continuamente funcionando, con lo que se tiene que prestar mucha atención a mecanismos como el de la recolección de basura (del inglés *garbage collection*) que liberen información de la memoria conforme ésta ya no se necesite. Además, hay que implementar técnicas que garanticen la tolerancia a fallos (como por ejemplo el uso de arquitecturas distribuidas y de múltiples sensores).

Además, es muy importante que se puedan realizar operaciones de mantenimiento on-line, tales como la incorporación de nuevos módulos, corrección de errores en el software, actualización de versiones, etc... En determinadas aplicaciones, como por ejemplo el control de un horno de cemento, de una central nuclear o de un alto horno,

una parada del sistema puede tener consecuencias catastróficas e inutilizar la planta, ocasionado un grave perjuicio económico y generando altos riesgos ambientales.

14.7.7 Datos inciertos o imprecisos

Como ya se ha dicho, la validez de los datos tiende a decaer con el paso del tiempo. Además, su validez puede ser cuestionable debido a una degradación en el funcionamiento de los sensores, como puede ser el caso de un entorno con elevado ruido o en el que los sensores están expuestos a alteraciones por condiciones meteorológicas adversas o por acumulación de suciedad. Este problema hace necesario que un STRBC sea capaz de detectar y tratar de manera apropiada los datos inciertos.

La imprecisión de los datos es otro problema a tener en cuenta. Sensores de baja calidad o que se han degradado con el paso del tiempo, pueden proporcionar medidas sujetas a un intervalo de error. Su información debe ser tratada como imprecisa y no como información exacta.

Es necesario modelar explícitamente la imprecisión e incertidumbre asociada a los datos mediante técnicas apropiadas: probabilidades, lógica difusa, etc. Las técnicas de fusión de datos son otra valiosa herramienta para mejorar la fiabilidad y la precisión. Así, por ejemplo, si en una aplicación de monitorización en medicina perdemos la información de frecuencia respiratoria proporcionada por el correspondiente sensor, podemos extraer esta misma información de otra fuente, procesando la señal electrocardiográfica recogida por otros sensores. Esta señal está sometida a una modulación de baja frecuencia producida por el movimiento de los músculos que intervienen en la respiración, y mediante un sencillo procesamiento permite calcular la frecuencia respiratoria a partir de esta fuente de datos alternativa.

Por otra parte, el sistema tiene que ser capaz de desempeñar su función aunque en un momento dado no tenga disponible todos los datos que necesita. Esto obliga a la implementación de mecanismos de razonamiento que sean capaces de alcanzar conclusiones en ausencia de algunos datos y sin tener que preguntar al usuario por ellos, teniendo en cuenta que los razonamientos serían más precisos en presencia de dichos datos. En [Palma y otros, 1998] se presenta un sistema de diagnóstico abductivo en tiempo real, que es capaz de producir respuestas aun en ausencia de algunos datos importantes. Cuando el sistema necesita un dato cuyo valor es desconocido, las respuestas que da son más imprecisas, pudiendo ser refinada dicha respuesta una vez que los valores desconocidos se introducen en el sistema. Esto se consigue gracias a que el mecanismo de razonamiento abductivo implementado en ese sistema es capaz de incorporar información de forma retrospectiva.

14.7.8 Razonamiento temporal

El tiempo es la variable más importante en un STRBC. Los datos con los que trabaja un STRBC cambian continuamente en el tiempo. En muchas de sus aplicaciones, es necesario tener en cuenta los valores pasados de los datos para poder establecer conclusiones en el tiempo actual. En ocasiones, también es necesario hacer predicciones en tiempo futuro, comprobando posteriormente si se satisfacen.

Las herramientas para desarrollo de STRBC suelen incluir alguna técnica de *razonamiento temporal*, en general bastante simple. Estas técnicas proporcionan la posibilidad de representar hechos temporales, en donde cada valor está asociado a su tiempo de creación. Proporcionan además la posibilidad de representar y aplicar conocimiento temporal. Para ello, el lenguaje de representación de la herramienta permite expresar condiciones sobre valores previos y aplicar operadores temporales, como promedios, tendencias, etc.

14.7.9 Tiempo de respuesta garantizado

Un STRBC debe ser capaz de responder dentro de un período de tiempo predeterminado, es decir, el sistema debe producir una respuesta lo más satisfactoria posible antes de que se cumplan los tiempos de respuesta límites. Esto es lógico si se tiene en cuenta que los STRBC operan en entornos dinámicos, y por tanto, las decisiones que se tomen tienen que ser aplicadas antes de que cambien las condiciones que dieron lugar a ellas.

Se emplean técnicas de *Razonamiento en Tiempo Crítico*. A continuación se citan algunas de las técnicas para este tipo de razonamiento, aunque un análisis más detallado se puede encontrar en [Vivancos y otros, 1997] y [Garvey y Lesser, 1994].

- Razonamiento Progresivo: La Base de Conocimiento se organiza en capas sucesivas, donde cada capa contiene un conocimiento más completo que la anterior, y por lo tanto permite una respuesta más precisa. De esta forma, el mecanismo de inferencias se concentra en la primera capa y obtiene una respuesta aproximada. Si el tiempo consumido es menor que el tiempo límite, el mecanismo de inferencia aplica la segunda capa, refinando de esta forma la respuesta obtenida. El sistema continúa aplicando capas hasta que el tiempo disponible se agote.
- Red de Inferencias con Restricciones Temporales: Se basa en la definición de una estrategia de inferencia diseñada para trabajar en entornos de trabajo limitados en tiempo. La estrategia de inferencia considera primero el conocimiento más relevante. Los datos menos significativos serán tenidos en cuenta dependiendo de la disponibilidad de tiempo para su procesamiento. Se utiliza una red semántica para representar el conocimiento sobre el dominio y restricciones temporales para guiar la búsqueda.
- Lógica de Precisión Variable: La lógica de precisión variable permite resolver problemas de razonamiento con información incompleta y restricciones en los recursos. Además, ofrece mecanismos para alcanzar un compromiso entre la precisión de las inferencias y la eficiencia computacional en su obtención. Los dos aspectos que se consideran en la solución son la especificidad de la respuesta y la credibilidad de la misma. Este método se basa en la presentación de unas reglas de producción censuradas, que son reglas que incluyen una parte que sólo se evalúa si hay tiempo suficiente. De esta forma, el sistema puede encontrar una solución de compromiso entre la precisión de la respuesta y el tiempo que se tarda en obtenerla.

- Algoritmos Anytime: Un algoritmo anytime construye la respuesta mediante un proceso de refinamiento iterativo que puede ser interrumpido en cualquier momento. Obviamente, la respuesta será más precisa cuanto más tiempo esté ejecutándose el algoritmo. La gran ventaja de este tipo de algoritmos es que siempre dan una respuesta, con lo que siempre pueden responder a los cambios del entorno. Basándose en este tipo de algoritmos se han desarrollado otros métodos como la Computación Imprecisa, donde se incluye en cada tarea una parte obligatoria que tiene que ejecutarse de forma ininterrumpida para garantizar la validez de la respuesta.

14.7.10 Foco de atención

Cuando aparece un evento que reviste cierta importancia, un STRBC debe poder concentrar los recursos necesarios en la obtención de una respuesta al mismo. Este cambio de foco de atención puede implicar la activación de un nuevo módulo de conocimiento que se encargue de modificar el conjunto de sensores a los que está atendiendo el sistema, y posiblemente la frecuencia de muestreo aplicada a dichos sensores. Esto se puede conseguir mediante la definición de una serie de contextos en los que se pueden aplicar determinados módulos de conocimiento, de tal forma que dichos contextos se activan a medida que los eventos van llegando.

Esa característica también implica la necesidad de un razonamiento de tipo oportunista que sea capaz de elegir la mejor acción en respuesta a una situación concreta. Ello requiere un mecanismo de control de tipo deliberativo, capaz de evaluar las prioridades de atención asociadas a la nueva información y la utilidad de las posibles acciones que se pueden aplicar.

14.7.11 Herramientas para Sistemas Expertos en Tiempo Real

14.7.11.1 Introducción a G2

G2 de la compañía Gensym Corp. (www.gensym.com) es una herramienta para el desarrollo rápido de prototipos e implementación de STRBC. Dispone de un entorno de desarrollo gráfico y orientado a objetos, con todas las características deseables para un entorno de desarrollo y ejecución de STRBC [Hangos y otros, 2001]. Los componentes básicos de G2 son: el Mecanismo de Inferencias (MI), la Base de Conocimiento (BC), la Base de Datos Temporal (BDT), y la Interfaz de Usuario (IU). Además, proporciona numerosas extensiones, como:

- *Telewindows*, permite la ejecución en más de un puesto de trabajo.
- *G2 Gateway standard interface (GSI)* , permite la comunicación con sistemas externos como bases de datos, aplicaciones C/C++, y software de simulación externo.
- *NeurOn-Line*, para desarrollo de redes neuronales.

- *G2 Weblink*, permite el acceso a aplicaciones en G2 mediante un explorador Web.
- *G2 Javalink*, permite integrar aplicaciones en G2 con Java.

G2 ofrece una serie de ventajas respecto a otras herramientas de desarrollo de STRBC, como RtWorks (www.talarian.com), ya que minimiza el tiempo invertido en el desarrollo de un primer prototipo y ofrece mayor variedad de utilidades software (proporcionadas por el fabricante dada la fuerte dependencia entre módulos). G2 permite obtener un resultado optimizado en poco tiempo cuando los requerimientos de la aplicación son moderados y pueden ser satisfechos con las capacidades de la herramienta, lo que se traduce en menores costes de desarrollo.

14.7.11.2 Representación del conocimiento

El estado del problema se representa mediante una taxonomía de clases, definidas por una serie de propiedades (atributos) comunes a todas las instancias (objetos) de la clase. Cada atributo de una clase tiene un tipo asociado, y representa una variable que tomará valores a lo largo del tiempo. Para cada atributo se puede configurar el número de valores históricos del atributo que se van a almacenar en la memoria del sistema, así como el intervalo de validez del valor, que especifica con qué intervalo de tiempo se debe actualizar el valor de la variable.

La base de conocimiento de G2 admite la creación de reglas y de procedimientos. G2 ofrece cinco tipos de reglas [Gensym, 2004]:

- Reglas de producción del tipo *if <condiciones>then <acciones>[else <acciones>]*
- Reglas *initially* que se ejecutan cuando un *workspace* es activado.
- Reglas *unconditionally* que ejecutan su acción siempre que son invocadas.
- Reglas *when* que son similares a las reglas *if*, pero no pueden participar en el encadenamiento del motor de inferencia.
- Reglas *whenever* que se ejecutan solamente cuando tienen lugar determinados eventos.

Además, se distinguen entre reglas específicas, que se aplican a valores u objetos específicos, y reglas genéricas, que se aplican a un conjunto de valores u objetos mediante el prefijo *for*.

G2 invoca a las reglas mediante diversos métodos, entre los que se encuentran el encadenamiento hacia delante y hacia atrás, la detección de eventos, los intervalos de tiempo específicos, al activar workspaces, y focalizando la atención en elementos o en categorías de reglas. Incluso se puede especificar si una regla ejecuta sus acciones secuencialmente o en paralelo. Además, G2 permite la definición de procedimientos y métodos que se utilizan para realizar cálculos que forman parte de las condiciones o las acciones de las reglas de producción (calcular tendencias, construir un mensaje al usuario, aplicar métodos de resolución de ecuaciones diferenciales, ...).

14.7.11.3 Estrategias de ejecución de las reglas

G2 permite el uso de estrategias para focalizar la atención del motor de inferencia sobre una zona específica de la base de conocimiento, y permitir la ejecución eficiente. Para establecer estas estrategias, toda regla posee un conjunto de atributos que permiten establecer los distintos modos de operación, entre los que se encuentran:

- Período de Ejecución de una Regla (*Scan Interval*).
- Agrupación de reglas.
- Objetos y Clases focales (*Focal Objects/Classes*).

El período de ejecución de una regla (*Scan Interval*) es un valor que se puede asociar a una regla para indicar cada cuanto tiempo el motor de inferencia debe aplicar la regla.

G2 permite modularizar la base de conocimiento en conjuntos de reglas dedicadas a tareas específicas, por ejemplo gestión de alarmas, acción sobre los efectores, actualización de parámetros de control, generación de consejos, etc... Cuando hay que ejecutar una de estas tarea, el motor de inferencia selecciona la categoría correspondiente y aplica sólo las reglas incluidas en la categoría, es decir, aquellas que han sido definidas asignando un valor al atributo *categories* de la regla, indicando en qué agrupación va a estar.

La definición de objetos y clases focales (*Focal Objects Classes*) es otro método para modularizar la base de conocimiento: consiste en asociar cada elemento de la base de datos temporal con un conjunto de reglas que el motor de inferencia chequeará cuando detecte alguna situación de control referida a ese elemento.

14.7.11.4 Razonamiento temporal

G2 permite referenciar los valores de las variables en un cierto tiempo (actual o pasado). Para obtener el tiempo actual del sistema se usa la expresión *the current time*. Para hacer referencia al valor de una variable en un cierto tiempo se utiliza la expresión:

the value of <variable>as of { <time-interval>|<number>datapoints } ago donde:

<time-interval>son las unidades temporales que retrocedemos desde el instante actual.

<number>es número de valores que retrocedemos.

14.8 Lecturas recomendadas

Este es un listado de algunas lecturas complementarias relativas al control intelectual.

- von Altrock, C. (1995). *Fuzzy Logic & NeuroFuzzy Applications Explained* Estados Unidos de América: Prentice Hall.

- Colomer J., Meléndez J. y Ayza J. (2000). *Sistemas de Supervisión*. España: Cuadernos CEA-IFAC.
- Hangos, K. M.(2001). *Intelligent Control Systems: An Introduction with Examples* Estados Unidos de América: Kluwer Academic Publishers.
- Panos J. Antsaklis y Kevin M. Passino (1992). *An Introduction to Intelligent and Autonomous Control*. Estados Unidos de América: Kluwer Academic Publishers.

En la siguiente dirección se encuentran disponibles otros recursos relativos al control inteligente.

- **Sistemas de Control Inteligente, una introducción con ejemplos** (del inglés *Intelligent control systems an introduction with examples*) por Katalin M. Hangos <http://site.ebrary.com/lib/bual/Doc?id=10067285&ppg=244>

14.9 Resumen

En este capítulo se ha estudiado el control de procesos como una tarea intensiva en conocimiento. El capítulo se inicia con una introducción al problema, una revisión de conceptos básicos de control desde el punto de vista de la ingeniería y un breve análisis de los sistemas de control convencionales. Una vez presentado el problema, este capítulo aborda el control inteligente desde tres niveles diferenciados: *Nivel Metodológico*, *Nivel de Técnicas* y *Nivel de Diseño*.

En el *Nivel Metodológico* se considera el control como una tarea genérica, en el sentido de la Ingeniería del Conocimiento, estableciendo por qué y cuándo resulta conveniente la aplicación de técnicas de control inteligente frente a métodos convencionales de control. Así, se describen algunas propuestas de métodos de resolución de problemas para tareas de control en el marco de la *metodología CommonKADS*. En primer lugar se analiza la tarea de Control Supervisor, diferenciando entre las tareas de análisis de valores fuera de los rangos esperados (monitorización) y la actuación mediante órdenes para rectificar los valores inesperados (comandos de control). Finalmente se proponen una serie de plantillas para la resolución de las subtareas que pueden estar implicadas en un problema de control inteligente.

Para el *Nivel de Técnicas*, se ha elegido el *control borroso* por su alto impacto en soluciones para el mundo industrial y en dispositivos de electrónica doméstica. La gran diferencia respecto al control convencional es la posibilidad de definir estas funciones de control mediante conjuntos de reglas de producción, lo que permite modelar de forma relativamente simple el conocimiento de los expertos en supervisión y control de procesos, algo particularmente útil en aplicaciones complejas. Estas reglas de actuación son especificadas mediante descriptores lingüísticos. Así, se pueden usar términos tales como *muy caluroso* o *poco rápido* que representan información cualitativa poco precisa, pero que son un elemento sustancial del conocimiento humano. Para el desarrollo de un controlador borroso es necesario especificar: las variables borrosas de entrada y de salida, el método de fuzzificación, la base de reglas borrosas, el mecanismo de inferencias y el método de defuzzificación.

En el *Nivel de Diseño* se han estudiado arquitecturas y herramientas que soporan y facilitan esta última fase del desarrollo. Existen abundantes herramientas para el diseño e implementación de sistemas de control borroso, incluyendo algunas que facilitan las implementaciones hardware. Por su generalidad, facilidad de uso y disponibilidad, se ha elegido como soporte básico MATLAB y sus librerías *Simulink* y *Fuzzy Toolbox*.

Como último apartado este capítulo incluye un análisis de otros dos aspectos esenciales en muchas aplicaciones de control inteligente: el razonamiento reactivo y el razonamiento en tiempo real. En el campo del razonamiento reactivo se han analizado las *arquitecturas de pizarra*, estudiando la metáfora de la pizarra de Newell, sus componentes fundamentales y las fases del ciclo de control. Finalmente, en el campo del razonamiento en tiempo real se han analizado algunas de los aspectos fundamentales: los eventos asíncronos, el razonamiento no monótono, el modo continuado de operación, el razonamiento temporal, el procesamiento de datos imprecisos, el tiempo de respuesta garantizado y el cambio del foco de atención. Finalmente se hace una breve introducción a una herramienta para desarrollo de sistemas en tiempo real basados en conocimiento.

14.10 Ejercicios propuestos

14.1. Una *grúa gantry móvil* es un tipo particular de grúa puente que tiene una estructura horizontal sustentada por dos columnas verticales. Su objetivo es desplazar un objeto sobre el eje horizontal. El objeto se encuentra suspendido verticalmente, sujeto a un carro móvil que se desplaza horizontalmente a través de una correa accionada por un motor. El movimiento del carro desplaza al objeto de forma horizontal, sin embargo también producirá un movimiento oscilatorio cuando el carro móvil cambia su velocidad, arranca o se detiene. El objeto se comporta como un péndulo cuyo eje se desplaza horizontalmente.

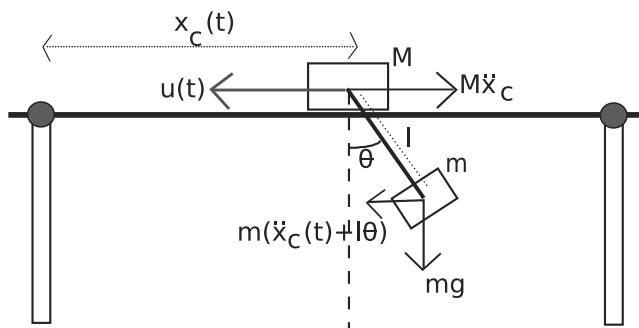


Figura 14.33: Diagrama de fuerzas de la *grúa gantry móvil*

El objetivo del ejercicio es diseñar un sistema controlador que haga que las oscilaciones que se produzcan sean mínimas. Se supone que existen unos sensores capaces de indicar la posición del carro y el ángulo de oscilación en cada momento. La señal controlada será la tensión eléctrica aplicada al motor que desplaza el carro.

Con el objeto de simplificar el problema, se describe a continuación el modelo físico del fenómeno¹¹. En primer lugar se describen de forma simplificada las fuerzas que actúan sobre el sistema (véase la Figura 14.33).

Se asume que el carro se desplaza sobre el puente sin fricción y que la carga, al oscilar, tampoco presenta ningún tipo de resistencia. Denominamos¹² $x_c(t)$ a la posición del carro en el instante t , $\theta(t)$ al ángulo de oscilación, y $u(t)$ a la fuerza aplicada por el motor eléctrico. La fuerza de inercia del carro es $M\ddot{x}_c(t)$, mientras que, para la carga, la fuerza gravitatoria es mg y la de inercia es $m(\ddot{x}_c(t) + l\ddot{\theta}(t))$. La suma de fuerzas horizontales y la suma de momentos debe ser nula, por lo que se satisface:

$$\begin{aligned}\sum F_x &= 0 \\ M\ddot{x}_c(t) + m(\ddot{x}_c(t) + l\ddot{\theta}(t)) &= u \sum M_Q = 0 \\ m(\ddot{x}_c(t) + l\ddot{\theta}(t))l\cos(\theta(t)) + mg\sin(\theta(t)) &= 0\end{aligned}$$

Para simplificar, se puede suponer que, cuando $\theta < 1$ (pequeñas oscilaciones), $\cos(\theta(t)) = 0$ y $\sin(\theta(t)) = \theta(t)$, por lo que la última expresión puede reducirse a:

$$m\ddot{x}_c(t) + ml\ddot{\theta}(t) + mg\theta(t) = 0 \quad (14.42)$$

Ambas ecuaciones describen el comportamiento de $x_c(t)$ y $\theta(t)$ y se pueden representar mediante la siguiente ecuación matricial:

$$\begin{bmatrix} M+m & ml \\ m & ml \end{bmatrix} \begin{bmatrix} \ddot{x}_c \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & mg \end{bmatrix} \begin{bmatrix} x_c \\ \theta \end{bmatrix} = \begin{bmatrix} u \\ 0 \end{bmatrix}$$

Despejando las incógnitas \ddot{x}_c y $\ddot{\theta}$, se obtiene:

$$\begin{bmatrix} \ddot{x}_c \\ \ddot{\theta} \end{bmatrix} = \frac{1}{ml(M+m)-m^2l} \begin{bmatrix} ml & -ml \\ -m & M+m \end{bmatrix} \begin{bmatrix} u \\ -mg\theta \end{bmatrix}$$

- Realice en *Simulink* de MATLAB una simulación del sistema, a partir del modelo matemático descrito anteriormente, dando valores concretos a m , M , l , g y u , siendo $m < M$ y pudiendo aproximar g a 10.
- Describa cuáles serían las entradas y salidas de un hipotético controlador borroso, indicando las etiquetas lingüísticas que se usarían.

¹¹Este estudio es un fragmento del trabajo realizado por Clark R. E. y Gavin M. Sin de la Universidad de Duke titulado *Gantry Crane: Analysis, Visualization, Experimentation and Control* y disponible en <http://weave.duke.edu/weave/>

¹²En esta notación, dado $z(t)$ variable respecto al tiempo, \dot{z} es la primera derivada respecto al tiempo.

3. Describa algunas reglas borrosas sencillas para el sistema de control.
4. Implemente el sistema borroso descrito mediante la *Fuzzy Logic Toolbox* de MATLAB.
5. Realice simulaciones del sistema controlado por el controlador implementado y añada las reglas convenientes para mejorar el sistema de control.

Referencias

- ANTSAKLIS, PANOS J. y PASSINO, KEVIN M.: *An Introduction to Intelligent and Autonomous Control*. Kluwer Academic Publishers, 1992. ISBN 0-7923-9267-1.
- BURNS, A. y WELLINGS, A.: *Real-Time Systems and Programming Languages*. Addison-Wesley, 1997.
- CARVER, N.: «A Revisionist View of Blackboard Systems». En: *Proceedings of the 1997 Midwest Artificial Intelligence and Cognitive Science Society Conference*, Dayton, Ohio, 1997.
- CHAU, K.W. y ALBERMANI, F.: «Hybrid knowledge representation in a blackboard KBS for liquid retaining structure design». *Engineering Applications of Artificial Intelligence*, 2004, **17**, pp. 11–18.
- COLOMER, JUAN; MELÉNDEZ, JOAQUIM y AYZA, JORDI: *Sistemas de Supervisión*. Cuadernos CEA-IFAC, España, 2000.
- CORKILL, D.: «Blackboard Systems». *AI Expert*, 1991, **6(9)**, pp. 40–47.
- CORKILL, D.: «Collaborating Software: Blackboard and Multi-Agent Systems and the Future». En: *Proceedings of the International Lisp Conference*, New York, 2003.
- CRAIG, I. D.: *Blackboard Systems*. Ablex, Norwood, NJ, 1994.
- ERMAN, L. D.; HAYES-ROTH, F.; LESSER, V. R. y REDDY, D. R.: «The Hearsay-II Speech Understanding System: Integrating knowledge to resolve uncertainty». *ACM Computing Survey*, 1980, **12**, pp. 213–253.
- GARVEY, A. y LESSER, V.: «A Survey of Research in Deliberative Real-Time Artificial Intelligence». *Journal of Real-Time Systems*, 1994, **6(3)**, pp. 317–347.
- GENSYM: *G2 Reference Manual, Version 8.0 Rev. 0*. Gensym Corporation, 2004.
- HANGOS, K.; LAKNER, R. y GERZSON, M.: *Intelligent Control Systems: An Introduction with Examples*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.
- HAYES-ROTH, B.: «A blackboard architecture for control». *Artificial Intelligence*, 1985, **26**, pp. 251–321.
- HAYES-ROTH, B. y HAYES-ROTH, F.: «A Cognitive Model of Planning». *Cognitive Science: A Multidisciplinary Journal*, 1979, **3(4)**, pp. 275–310.
- HINES, J. WESLEY: *Matlab supplement to Fuzzy and Neuronal Approaches in Engineering*. Jonh Wiley and Sons, Estados Unidos de Am*erica*, 1997.
- HUNT, J.: «Blackboard Architectures». *Informe técnico*, JayDee Technology Ltd., 2002.

- KOYMANS, R.: «Specifying Real-Time Properties with Metric Temporal Logic». *Journal of Real-Time Systems*, 1990, **2**(4), pp. 255–299.
- LAFFEY, T.; COX, A.; SCHMIDT, J.; KAI, S. y READ, J.: «Real-Time Knowledge-Based Systems». *AI Magazine*, 1988, **9**(1), pp. 27–45.
- METZNER, C.; CORTEZ, L. y CHACÓN, D.: «Using a Blackboard Architecture in a Web Application». En: *Proc. of Informing Science and Information Technology Education Joint Conference*, Flagstaff, USA, 2005.
- NEWELL, A.: «Some Problems of Basic Organization in Problem Solving Programs». En: Yovits; Jacobi y Goldstein (Eds.), *Proc. Conference on Self Organizing Systems*, Spartan Books, 1962.
- NG, GEE WAH: *Intelligent Systems - Fusion, Tracking and Control*. Research Studies Press, 2003. ISBN 0-86380-277-X.
- NII, H. E. y FEIGENBAUM, E. A.: «Rule-based understanding of signals». En: A. Waterman y E. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*, Academic Press, 1978.
- NII, H. P. y AIELLO, N.: «A knowledge-based program for building knowledge-based programs». En: *Proc. of Sixth International Joint Conference on Artificial Intelligence*, pp. 645–655, 1979.
- OH, K.W. y QUEK, C.: «BBIPS: a blackboard-based integrated process supervision». *Engineering Applications of Artificial Intelligence*, 2001, **14**, pp. 703–714.
- PALMA, J.; MARÍN, R.; SÁNCHEZ, J. L. y CÁRDENAS, M. A.: «A Diagnosis Task in an Intelligent Patient Supervisory System». En: José Cuena (Ed.), *Proceedings of the XV IFIP World Computer Congress*, pp. 159–172, 1998.
- PUIG ADAM, P: *Ecuaciones Diferenciales: aplicado a la física y técnica*. Biblioteca Matemática, Norwood, NJ, 1964.
- RANDELL, B.; LAPRIE, J.; KOPETZ, H. y LITTLEWOOD, B.: *Predictably Dependable Computing Systems*. Springer-Verlag, 1995.
- SEO, H.S. y CHO, T.H.: «An application of blackboard architecture for the coordination among the security systems». *Simulation Modelling Practice and Theory*, 2003, **11**, pp. 269–284.
- SIERRA, E. A.; QUIROGA, J. J.; FERNÁNDEZ, R. y MONTE, G. E.: «An intelligent maintenance system for earth-based failure analysis and self-repairing of microsatellites». *Acta Astronautica*, 2004, **55**, pp. 61–67.
- SIMMONS, GEORGE F.: *Ecuaciones Diferenciales*. Mc Graw Hill, España, 1993.
- SMITH, C. y CORRIPIO, A.: *Control Automático de Procesos*. Limusa, Mexico DF, 1994.

TERRY, A.: «The crysalis project: Hierarchical control of production systems». *Technical report hpp-83-19*, Stanford University, 1983.

VIVANCOS, E.; HERNÁNDEZ, L. y BOTTI, V.: «Técnicas y Arquitecturas de Inteligencia Artificial en Tiempo Real». *Informática y Automática*, 1997, **30**(4), pp. 5–22.

ZILOUCHIAN, ALI y JAMSHIDI, MO: *Intelligent Control Systems Using Soft Computing Methodology*. CRC Press, 2001. ISBN 0-8493-1875-0.

Parte V

APRENDIZAJE Y MINERÍA DE DATOS

Capítulo 15

Redes neuronales

Bertha Guijarro Berdiñas, Óscar Fontela Romero
y Noelia Sánchez Meroño
Universidad de A Coruña

15.1 Introducción

Uno de los problemas más antiguos de la ciencia experimental es encontrar funciones que ajusten, o expliquen, datos que se observan de fenómenos naturales. La principal ventaja de la existencia de tales funciones es la posibilidad de predecir el comportamiento del sistema natural en el futuro y controlar sus salidas mediante la aplicación de las entradas adecuadas. Algunos ejemplos interesantes podrían ser la predicción de valores en bolsa, la predicción meteorológica o la clasificación de formas tumorales. La dificultad estriba en que los datos observados tienden a ir acompañados de ruido, y los mecanismos exactos que los generan normalmente son desconocidos. En ocasiones será posible encontrar un modelo matemático exacto que explique el proceso del que provienen los datos que observamos. Muchas veces, sin embargo, no podremos dar detalles de ese proceso. El objetivo, en este caso, será *estimar* el modelo subyacente que genera los datos observados.

En estos casos, las técnicas de aprendizaje automático nos permiten establecer estos modelos utilizando datos de ejemplo o experiencias pasadas. De este modo se podrán identificar ciertos *patrones* o regularidades en los datos y así podremos construir buenas *aproximaciones* al problema. Dentro de estas técnicas, que podríamos llamar de estimación semiparamétrica, se encuentran las *Redes de Neuronas Artificiales* que se explicarán en este capítulo.

15.1.1 ¿Qué es una red de neuronas?

Una *Red de Neuronas Artificiales* (en adelante, RNA) es un paradigma de procesamiento de información inicialmente inspirado en el modo en el que lo hace el cerebro. El elemento clave de este paradigma es su estructura. Las RNA están compuestas por un cierto número de elementos de procesamiento o *neuronas* que trabajan al unísono

para resolver un problema específico. Las redes neuronales actuales se basan en el modelo matemático de neurona propuesto por McCulloch y Pitts en 1943 [McCulloch y Pitts, 1943]. En dicho modelo (véase Figura 15.1) cada neurona recibe un conjunto de entradas $\{x_1, x_2, \dots, x_D\}$ y devuelve una única salida y . Además, dentro de una RNA existen numerosas conexiones entre las distintas neuronas que la forman. Estas conexiones simulan las conexiones interneuronales del cerebro y, al igual que éstas, pueden establecerse con mayor o menor intensidad. En el caso de las RNA esta intensidad la determinan los *pesos sinápticos* (o simplemente *pesos*). De este modo, cada entrada x_i de una neurona se encuentra afectada por un peso w_i .

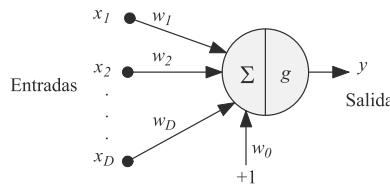


Figura 15.1: Modelo neuronal de McCulloch-Pitts.

El primer paso para obtener la salida y de la neurona es calcular la suma ponderada a de las entradas, llamada *activación* de la neurona:

$$a = \sum_{i=1}^D w_i x_i + w_0 \quad (15.1)$$

donde w_0 es un *umbral* o *sesgo* que se utiliza para compensar la diferencia entre el valor medio de las entradas, sobre todo el conjunto de entrenamiento, y el correspondiente valor medio de las salidas deseadas. Posteriormente, a partir de este valor a se obtiene la salida y de la neurona mediante la aplicación de una función, llamada *función de activación* o *de transferencia* $g(a)$, es decir:

$$y = g(a) = g \left(\sum_{i=1}^D w_i x_i + w_0 \right) = g \left(\sum_{i=0}^D w_i x_i \right) \quad (15.2)$$

donde, como se observa, es posible tratar el umbral w_0 como un peso más si se supone una entrada añadida x_0 con un valor fijo de 1 (véase Figura 15.1). Finalmente, también es posible reescribir esta ecuación en notación vectorial como $g(a) = g(\mathbf{w}^T \mathbf{x})$, si tomamos \mathbf{w} como el vector de pesos y \mathbf{x} como el vector de entradas a la red.

La función de transferencia empleada en este modelo básico de McCulloch-Pitts es la función *escalón* definida por la ecuación:

$$g(a) = \begin{cases} 0 & \text{cuando } a < 0 \\ 1 & \text{cuando } a > 0 \end{cases} \quad (15.3)$$

y que se puede ver en la Figura 15.2(a). Como veremos, en los modelos actuales se escogen otro tipo de funciones, normalmente monótonas y derivables. Las más comunes, cuyas formas se muestran también en la Figura 15.2, son la función lineal,

$$g(a) = a \quad (15.4)$$

la sigmoidea

$$g(a) = \frac{1}{1 + e^{-a}} \quad (15.5)$$

la tangente hiperbólica (\tanh)

$$g(a) = \tanh = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad (15.6)$$

y la gaussiana

$$g(a) = \exp\left(-\frac{(a - \mu)^2}{2\sigma^2}\right). \quad (15.7)$$

Al número de neuronas que componen una RNA y a la forma en cómo están conectadas entre sí, se le conoce como *topología* de la red. A lo largo de este capítulo veremos ciertos tipos de RNA que implican formas preestablecidas de ordenar y conectar las neuronas dentro de la red.

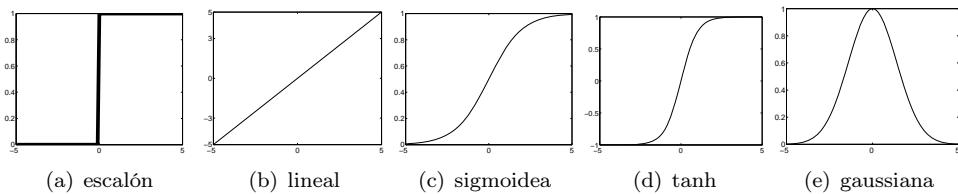


Figura 15.2: Principales funciones de transferencia utilizadas.

15.1.2 Tipos básicos de problemas

Existen dos tipos básicos de problemas que una RNA puede tratar de resolver: problemas de clasificación y problemas de regresión. En el primer tipo de problemas, el objetivo consiste en crear un procedimiento mediante el cual un nuevo caso representado por unos atributos observados o *características* que constituyen los datos de entrada a la RNA, se asigne a una de entre un conjunto de clases predefinidas. A la construcción de estos clasificadores también se le conoce como *reconocimiento de patrones*. Un ejemplo de problemas de este tipo sería el reconocimiento de rostros. En este caso, la entrada a la RNA es una imagen, las clases son las personas a reconocer, y el programa de aprendizaje deberá aprender a asociar imágenes de caras a las identidades.

Por otro lado, los problemas de regresión son problemas de *ajuste de funciones*. Es decir, se trata de obtener un número en función de los atributos de entrada a la red o, lo que es lo mismo, se trata de encontrar una función continua de ciertas variables. Un ejemplo de este tipo de problemas sería, por ejemplo, poder predecir la evolución de los consumos energéticos de un país a lo largo del año en función de ciertos datos históricos, época del año, etcétera.

15.1.3 Proceso de entrenamiento o aprendizaje

Como hemos visto, las RNA reciben unos datos de entrada que se transforman para producir una salida, con el objetivo de clasificarlos o ajustar una función. La justificación teórica para este tipo de aplicaciones es que, suponiendo que la red tenga un número suficiente de neuronas, podrá ajustar cualquier función continua con una cierta precisión con tan sólo escoger los valores adecuados para los parámetros ajustables de dicha red. Estos parámetros son, en general, los pesos sinápticos y son, por tanto, el medio que la red emplea para almacenar su conocimiento sobre el problema a resolver.

Este conocimiento se almacena en la RNA a través de un proceso de aprendizaje o *entrenamiento*, que no es más que la modificación de los parámetros de la RNA mediante un procedimiento preestablecido, al objeto de conseguir una mejora en su rendimiento. Como el proceso de aprendizaje humano, el de las RNA está basado en el uso de ejemplos que representan el problema. A este conjunto se le conoce como *conjunto de entrenamiento*. Es importante recalcar que el objetivo del aprendizaje no es memorizar las relaciones entrada/salida que hay en el conjunto de entrenamiento, sino modelar el proceso que ha generado estos datos. Para ello es conveniente que el número y tipo de ejemplos disponibles para el entrenamiento de la red sea suficientemente representativo de la relación que se desea aprender. De este modo, una vez entrenada, la red será capaz de manejar no sólo los datos de entrenamiento, sino nuevos datos distintos de los primeros, sin que por ello se degrade su rendimiento. Esto se conoce como la capacidad de *generalización* de la red.

En función de cómo esté constituido el conjunto de entrenamiento se distinguen básicamente dos tipos de aprendizaje: supervisado y no supervisado. En el *aprendizaje supervisado* se parte de una serie de observaciones o *entradas* y unas *salidas deseadas* que la red debería obtener, y el objetivo es aprender la correspondencia entre ambas. Es decir, el conjunto de entrenamiento estará compuesto por pares de la forma:

$$\mathbf{X} = \{(\mathbf{x}^n, \mathbf{t}^n)\}_{n=1}^N \quad (15.8)$$

donde \mathbf{x} es el vector de entradas, \mathbf{t} el de salidas deseadas y N es el tamaño del conjunto de entrenamiento. La aproximación que se sigue es partir del modelo genérico $f(\cdot)$ que representa la red, totalmente definido excepto por una serie de parámetros o pesos \mathbf{w} , y con él producir una buena aproximación a \mathbf{t} :

$$y = f(\mathbf{x} \mid \mathbf{w}) \quad (15.9)$$

donde y representa la salida proporcionada por la red. El algoritmo de aprendizaje optimizará los parámetros de manera que la salida producida por el modelo sea lo más parecida posible a la del conjunto de entrenamiento. Es decir, se trata de encontrar el conjunto de parámetros w^* que *minimiza el error* de aproximación. Es, por tanto, necesario definir una función de error E que, en cada paso del aprendizaje, indique lo cerca que se está de la solución.

De nuevo existe una serie de funciones de error que, por sus características, se utilizan más frecuentemente. Todas ellas toman la forma de una suma del error que, para cada ejemplo n de aprendizaje, mide la diferencia entre la salida y^n obtenida por la red y la salida deseada t^n que se debería haber obtenido. La forma de medir dicho error dependerá del tipo de problema que se intente resolver. Así, si se trata de un problema de *regresión*, la salida de la red es una variable continua y la función de error más utilizada es el *Error Cuadrático Medio* (en adelante, ECM):

$$E = \frac{1}{N} \sum_{n=1}^N (y^n - t^n)^2. \quad (15.10)$$

El uso de esta función permitirá que las salidas reales de la red modelen la función de distribución media de las salidas deseadas utilizadas durante el entrenamiento.

Por otro lado, si hablamos de problemas de *clasificación*, la salida de la red no será única sino que deberá estar formada por tantas neuronas como clases posibles C , y la salida deseada estará codificada como un vector $\mathbf{t} = (t_1, t_2, \dots, t_C)$ de valores binarios $(0, 1)$ en el que sólo el bit correspondiente a la clase adecuada tomará el valor 1. En este caso, el objetivo es conseguir que las salidas de la red modelen las probabilidades de pertenencia de las entradas a alguna de las clases consideradas. Para conseguirlo, en esta situación resulta más adecuado utilizar la función de *entropía cruzada*, cuya ecuación es

$$E = - \sum_{n=1}^N \sum_{k=1}^C (y_k^n) t_k^n \quad (15.11)$$

donde y_k^n es la salida de la neurona k para el ejemplo de entrenamiento n . La diferencia fundamental respecto al *ECM* es que, en este caso, no se valorará tanto la diferencia exacta entre el valor deseado t_k^n para cada neurona k de salida y el obtenido y_k^n , sino que el resultado de la clasificación sea el correcto. Es decir, que la neurona que representa a la clase correcta sea la que obtenga el mayor valor de activación a su salida.

Por otro lado, el tipo de problema y la función de error utilizada están muy ligados al tipo de función de activación empleada en las neuronas de salida. En el caso de problemas de clasificación, lo ideal es obtener valores discretos en las neuronas de salida (p.e., $(0, 1)$) y que, además, éstas se puedan interpretar como probabilidades para lo que, en conjunto, la suma de todas ellas deberá estar en el intervalo $[0, 1]$. Con este fin, es recomendable utilizar la función sigmoidea descrita en la ecuación (15.5). Una exposición interesante sobre la adecuación y características de diversas funciones de error puede encontrarse en [Bishop, 1995].

Otra forma alternativa de entrenamiento es el *aprendizaje no supervisado*. En este caso, en el conjunto de entrenamiento los datos de entrada \mathbf{x} no vienen acompañados

de una *salida deseada*, y el objetivo no es encontrar un mapeo entrada-salida, sino encontrar patrones que definen regularidades subyacentes en \mathbf{x} . Además, estos patrones ocurren con distintas frecuencias y la red deberá encontrar lo que ocurre con más generalidad. En algunos casos se tratará de encontrar una estimación de la *distribución de probabilidad* de \mathbf{x} , y en otros el objetivo será inferir las clases existentes en las que agrupar los ejemplos de entrada que presenten similitudes.

15.2 Métodos de aprendizaje supervisado

En esta sección se presentan las arquitecturas más comunes utilizadas en redes de neuronas basadas en aprendizaje supervisado. Como hemos visto, este tipo de aprendizaje se utiliza para modelar la correspondencia entre un conjunto de N ejemplos o patrones de entrada $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^N\}$ y las salidas deseadas $\{\mathbf{t}^1, \mathbf{t}^2, \dots, \mathbf{t}^N\}$, minimizando para ello la función de error que mide la diferencia existente entre estas salidas y las obtenidas por la red $\{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^N\}$.

15.2.1 Redes de neuronas de una capa: el Perceptrón

El *Perceptrón* es la red de neuronas artificiales más sencilla. Está compuesta únicamente por una capa de neuronas de entrada y otra capa de neuronas de salida. Para simplificar, en esta sección se considerará una única salida. Si bien este tipo de red apenas se emplea en la actualidad por las limitaciones que presenta, su estudio es obligado dado que es la base de los métodos actuales.

15.2.1.1 La regla de Hebb

En el modelo de McCulloch-Pitts (Figura 15.1) presentado previamente puede verse que la activación a de la neurona depende del valor que tomen las entradas y sus respectivos pesos, de forma que la variación de éstos origina distintas salidas para la misma entrada. Como ejemplo de funcionamiento de esta red, en la Tabla 15.1 se muestra la capacidad de este modelo para aprender las funciones lógicas AND y OR. Para ambas funciones, los pesos w_1 y w_2 tienen idéntico valor, basta variar el valor del umbral (peso w_0) para que dicho modelo represente una función u otra.

x_1	x_2	w_1	w_2	$\sum_{i=1}^2 w_i x_i$	AND	OR
0	0	1	1	0	0	0
0	1	1	1	1	0	1
1	0	1	1	1	0	1
1	1	1	1	2	1	1

Tabla 15.1: Simulación de las funciones AND y OR para el modelo de McCulloch-Pitts. El valor de w_0 es $-1,5$ para la función AND y $-0,5$ para la función OR.

Para que la red alcance el valor adecuado para los pesos, es necesario someterla a un entrenamiento o aprendizaje. Una de las primeras reglas de aprendizaje fue propuesta por D. Hebb en 1949 [Hebb, 1949], y se basa en un hecho biológico constatado: *cuando dos neuronas se activan simultáneamente su conexión se refuerza*. Esto es, cuando una neurona tiene un nivel positivo de actividad y se siente estimulada por otra neurona, tiende a producirse un refuerzo en la conexión que las enlaza. Por el contrario, si una neurona tiende a inhibir a otra, la fuerza de conexión entre ambas también tiende a disminuir. Suponiendo que en el modelo de la Figura 15.1, la entrada x_i es la salida de otra neurona, el peso w_i será incrementado cuando x_i e y sean positivos, pero también lo hará cuando ambos parámetros sean negativos. En contraposición, dicho peso se decrementará cuando x_i e y tengan signos contrarios. Esta idea se expresa matemáticamente como:

$$w_i(\tau + 1) = w_i(\tau) + \eta x_i y, \quad (15.12)$$

esto es, un peso w_i en un determinado instante del entrenamiento $\tau + 1$, es igual al peso en el instante anterior más un incremento o decremento. La magnitud de dicho cambio viene determinada tanto por la entrada x_i y la salida y de la neurona como por η conocida como ratio o *tasa de aprendizaje*. Esta es una constante positiva que determina en qué proporción se modifica el peso sináptico (cuanto mayor es la modificación) y, en cierto modo, controla la velocidad del proceso de aprendizaje. Se puede observar que esta regla de aprendizaje es *no supervisada*, ya que se emplea la salida de la neurona, y , pero no la salida deseada, t .

15.2.1.2 El Perceptrón

En 1957 Frank Rosenblatt, basándose en el modelo de McCulloch-Pitts y empleando como regla de aprendizaje una modificación de la propuesta por Hebb, presentó el perceptrón, el primer modelo de red de neuronas artificiales [Rosenblatt, 1962]. La arquitectura que Rosenblatt definió para el perceptrón consistía en una primera capa de j neuronas con funciones ϕ_j que se encargaban de transformar los datos de entrada. Estas funciones reciben un subconjunto aleatorio de entradas a través de unos pesos fijos y les aplican una función de activación de tipo escalón (véanse ecuación (15.3) y Figura 15.2(a)). De nuevo existe un peso especial o sesgo w_0 y, de igual modo que en la ecuación (15.2) se definía una entrada artificial x_0 con valor 1 asociado a este peso, en el caso del perceptrón se considera una función de activación extra $\phi_0 = 1$. La salida de esta primera capa de funciones ϕ_j se conecta a través de unos pesos a una última neurona para, finalmente, calcular la salida del perceptrón como:

$$y = g(a) = g\left(\sum_{j=0}^M w_j \phi_j(\mathbf{x})\right) = g(\mathbf{w}^T \boldsymbol{\phi}), \quad (15.13)$$

donde $\boldsymbol{\phi}$ es el vector formado por las funciones de activación ϕ_0, \dots, ϕ_M y g es la función escalón (véase ecuación (15.3)) definida para los valores $\{-1, 1\}$.

La principal aportación del perceptrón es que la adaptación de sus pesos se realiza teniendo en cuenta el error entre la salida que obtiene la red y la salida que se desearía

obtener. Es, por tanto, el primer método de aprendizaje supervisado. Para ilustrar este método, supóngase un problema de clasificación binaria (dos clases) en el que a cada vector de entrada \mathbf{x}^n se le asocia un valor objetivo t^n , de modo que la salida deseada para la red es $t^n = 1$, si \mathbf{x}^n pertenece a la clase \mathcal{C}_1 , y $t^n = -1$, si \mathbf{x}^n pertenece a la clase \mathcal{C}_2 . Teniendo en cuenta que cada vector de entrada \mathbf{x}^n genera su correspondiente vector de activación ϕ^n y considerando las ecuaciones (15.13) y (15.3) obtenemos que, para que el perceptrón realice una clasificación correcta, deberá cumplir que $\mathbf{w}^T \phi^n > 0$, si $\mathbf{x}^n \in \mathcal{C}_1$, y $\mathbf{w}^T \phi^n < 0$, si $\mathbf{x}^n \in \mathcal{C}_2$. O, lo que es lo mismo, $(\mathbf{w}^T \phi^n)t^n > 0$. Ello sugiere la minimización de la siguiente función de error conocida como el *criterio del perceptrón*:

$$E^{perc}(\mathbf{w}) = - \sum_{\phi^n \in \mathcal{M}} (\mathbf{w}^T \phi^n)t^n, \quad (15.14)$$

donde \mathcal{M} es el conjunto de vectores ϕ^n que han sido clasificados incorrectamente, de modo que E^{perc} será igual a 0 cuando todos los vectores han sido adecuadamente clasificados.

Con el objetivo de determinar los pesos \mathbf{w} que minimizan la función de error E se sigue un proceso adaptativo que consiste en comenzar con unos valores iniciales \mathbf{w} aleatorios e ir modificándolos iterativamente cuando la salida del perceptrón no coincide con la salida deseada. Así, cada vez que un patrón de entrada \mathbf{x}^n se clasifica incorrectamente, si $\mathbf{x}^n \in \mathcal{C}_1$ los pesos se incrementan y, por el contrario, se decrementan si $\mathbf{x}^n \in \mathcal{C}_1$ de acuerdo a la regla conocida con el nombre de *regla de aprendizaje del perceptrón simple*:

$$w_j(\tau + 1) = w_j(\tau) + \eta \phi_j^n t^n, \quad (15.15)$$

donde η representa la tasa de aprendizaje.

Conviene señalar que los perceptrones tienen varias limitaciones. Primero, los valores de salida del perceptrón sólo pueden ser binarios, debido a la función de activación empleada. Segundo, los perceptrones sólo pueden clasificar vectores linealmente separables, es decir, los pesos determinan zonas de clasificación separadas por un hiperplano. Así, en el caso de un perceptrón con 2 entradas, se clasificarán correctamente todos los patrones sólo si se pudiera establecer una línea recta entre las clases tal y como se aprecia en la Figura 15.13(a). Se puede comprobar fácilmente, por ejemplo, su incapacidad para aprender la operación XOR (O exclusivo) que implica datos no separables linealmente. Por otro lado, el teorema de la convergencia del perceptrón garantiza que, para cualquier conjunto de datos linealmente separable, la regla de aprendizaje expresada en la ecuación (15.15) finaliza en un número finito de pasos [Bishop, 1995; Haykin, 1999].

Otro modelo de neuronas similar al perceptrón es la *red lineal adaptativa* o ADALINE (*ADAptive LInear NEtwork*). La principal diferencia entre ambos modelos es la regla de aprendizaje que emplean éstas últimas, conocida como *regla Widrow-Hoff* [Widrow y Hoff, 1960], *regla LMS* (*Least Mean Squares*) o *regla delta* y representada como:

$$w_j(\tau + 1) = w_j(\tau) + \eta \delta^n \phi_j^n, \quad (15.16)$$

donde $\delta^n = y^n - t^n$. En este caso, las actualizaciones de los pesos son continuas y la regla se va acercando asintóticamente a la solución, mientras que en el perceptrón las actualizaciones son discretas y, por tanto, converge a la solución en un número finito de iteraciones.

15.2.2 El perceptrón multicapa

Se ha visto que el perceptrón simple es capaz de resolver problemas de clasificación e implementar funciones lógicas, como por ejemplo la función OR, pero es incapaz de implementar otras funciones simples como la función lógica XOR. Minsky y Papert publicaron un libro titulado *Perceptrons* [Minsky y Papert, 1969] en el que se exponían estas limitaciones, y que supuso el abandono por parte de muchos científicos de la investigación en redes neuronales, pues no se encontraba un algoritmo de aprendizaje capaz de implementar funciones de cualquier tipo. Las limitaciones de las redes de una sola capa hicieron que se plantease la necesidad de implementar redes en las que se aumentase el número de capas introduciendo capas intermedias entre la capa de entrada y la capa de salida, de manera que se pudiese implementar cualquier función con el grado de precisión deseado. La función que cumple dicha capa intermedia es tratar de realizar una proyección en la que resulten separables linealmente los patrones de entrada de manera que la unidad de salida pueda realizar una clasificación correcta. Surge así el perceptrón Multicapa, en adelante MLP (*MultiLayer Perceptron*), cuya arquitectura mostrada en la Figura 15.3, incluye una o varias capas intermedias de unidades procesadoras, también denominadas *capas ocultas* porque no tienen conexiones con el exterior.

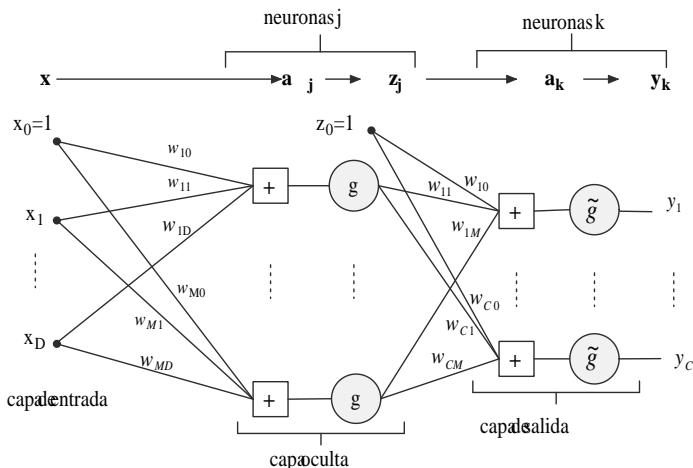


Figura 15.3: Arquitectura de un perceptrón multicapa con una capa oculta.

Aquí, simplemente consideraremos un MLP con una única capa oculta tal y como se representa en la Figura 15.3, si bien el hecho de añadir más capas sólo supone

repetir las operaciones que se van a exponer a continuación. Así, dicho MLP consta de D entradas, M neuronas en su capa oculta y C unidades de salida. El nivel de activación a_j de la neurona j de la capa oculta se calcula como una combinación lineal de las D entradas x_i que recibe sobre la que, tras aplicar una función de transferencia g se obtiene la salida z_j de dicha neurona:

$$z_j = g(a_j) = g\left(\sum_{i=0}^D w_{ji}x_i\right); \quad k = 1, 2, \dots, M, \quad (15.17)$$

donde w_{ji} es el peso asociado a la neurona j y la entrada x_i . De manera similar, cada salida de la red se obtiene como una suma ponderada de las salidas de las unidades de la capa oculta, sobre la que se aplica una función de transferencia, es decir, la salida de la neurona k viene dada por:

$$y_k = \tilde{g}(a_k) = \tilde{g}\left(\sum_{j=0}^M w_{kj}z_j\right) = \tilde{g}\left(\sum_{j=0}^M w_{kj}g\left(\sum_{i=0}^D w_{ji}x_i\right)\right); \quad k = 1, 2, \dots, C. \quad (15.18)$$

Es importante destacar que las funciones de transferencia g y \tilde{g} no tienen que ser iguales, por ello la notación que se emplea es diferente. Como veremos posteriormente, la función de transferencia g suele tomar la forma de las mostradas en la Figura 15.2, salvo la función escalón, mientras que la forma de la función \tilde{g} depende del tipo de problema, tal como se explica en la sección 15.1.3.

15.2.2.1 Algoritmo de retropropagación del error

Al igual que el perceptrón simple, el MLP basa el aprendizaje de sus pesos en una regla de ajuste del error, esto es, trata de determinar los pesos de las conexiones sinápticas de manera que las salidas de la red coincidan con las salidas deseadas, o por lo menos, sean lo más próximas posibles. Como veremos, el algoritmo de aprendizaje del MLP utiliza el método de descenso del gradiente para ajustar los pesos de la red. Además, dicho ajuste se realiza comenzando por la capa de salida, según el error cometido, y propagando este error a las capas anteriores, hasta llegar a la capa de las unidades de entrada, de ahí que se denomine *algoritmo de retropropagación del error* (*backpropagation algorithm*). Básicamente consiste en dos fases que se repiten hasta conseguir minimizar el error:

- En la primera fase, o paso hacia adelante, se aplica un patrón a las entradas de la red y su efecto se propaga a través de la misma, capa a capa. Finalmente, la red presenta un conjunto de salidas como respuesta a dicho patrón de entrada. Es decir, este paso consiste en el cálculo de la ecuación (15.18) para un conjunto de pesos dado. Normalmente, en la primera iteración del algoritmo se parte de un conjunto de pesos con un valor inicial pequeño aleatorio.
- En la siguiente fase, o paso hacia atrás, los pesos de la red se recalculan de acuerdo con una regla de ajuste del error. Esto es, se calcula el valor de la función

de error que compara la respuesta actual de la red y la respuesta deseada, y este error se propaga hacia atrás.

A continuación se explica este último paso en detalle. Para ello, en esta sección, se asumirá como función de error el ECM, una de las más ampliamente utilizadas. Así, para cada patrón \mathbf{x}^n se tiene la siguiente función de error:

$$E^n = \frac{1}{2} \sum_{k=0}^C (e_k^n)^2, \quad (15.19)$$

donde el factor $1/2$, que no altera el resultado de la función, se introduce para facilitar los cálculos posteriores, y el error e_k^n se define como:

$$e_k^n = y_k^n - t_k^n. \quad (15.20)$$

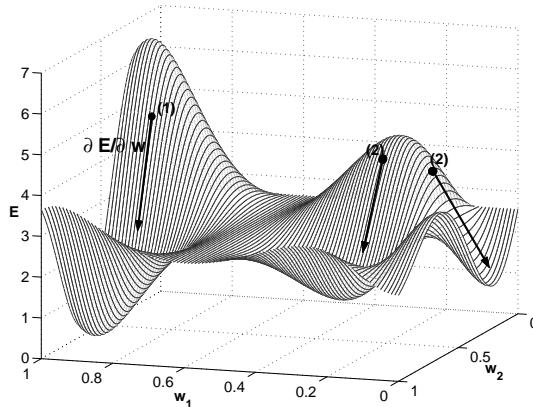


Figura 15.4: Ejemplo de descenso gradiente en una red con únicamente 2 pesos. Las flechas indican direcciones en el espacio de búsqueda suponiendo distintos valores iniciales para los pesos w_1 y w_2 . El punto (1) conduce a un mínimo global mientras que los puntos (2) conducen a mínimos locales.

Supongamos que nos encontramos en la iteración n del algoritmo de aprendizaje en el que se ha introducido un patrón \mathbf{x}^n ¹. Como se ha visto en las reglas de aprendizaje anteriores, el algoritmo de retropropagación corrige el peso sináptico w_{kj} añadiéndole un incremento Δw_{kj} . En este caso, el incremento es proporcional al gradiente $\partial E / \partial w_{kj}$ dado que, si consideramos la superficie de error que se forma al representar E junto el espacio de pesos, este gradiente determina la dirección de la búsqueda en este espacio para obtener el valor del peso w_{kj} que conduce a un mínimo de E . La Figura 15.4 muestra un ejemplo sencillo en el que la red sólo tiene dos pesos. Se

¹En lo sucesivo, para facilitar la legibilidad se prescindirá del superíndice n , asumiendo que todas las ecuaciones se refieren a resultados obtenidos para una entrada \mathbf{x}^n determinada.

observa que siguiendo la dirección indicada por el gradiente se conseguirá minimizar el error aunque no siempre se podrá alcanzar el resultado óptimo. De acuerdo con la regla de la cadena para derivadas parciales, este gradiente se representa como:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial e_k} \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} = -e_k \tilde{g}'(a_k) z_j \quad (15.21)$$

que se obtiene tras derivar las ecuaciones (15.19), (15.20), (15.18) y $a_k = \sum_{j=0}^M w_{kj} z_j$ con respecto a las variables indicadas, esto es:

$$\frac{\partial E}{\partial e_k} = e_k \quad \frac{\partial e_k}{\partial y_k} = -1 \quad \frac{\partial y_k}{\partial a_k} = \tilde{g}'(a_k) \quad \frac{\partial a_k}{\partial w_{kj}} = z_j. \quad (15.22)$$

Teniendo en cuenta que el *gradiente local* δ de una neurona k se define como:

$$\delta_k = \frac{\partial E}{\partial e_k} \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial a_k} = -e_k \tilde{g}'(a_k) \quad (15.23)$$

se tiene que el incremento que se debe aplicar al peso w_{kj} es:

$$\Delta w_{kj} = \eta \frac{\partial E}{\partial w_{kj}} = -\eta \delta_k z_j, \quad (15.24)$$

donde η es el ratio de aprendizaje. Esta ecuación señala que, para obtener el incremento necesario, únicamente es necesario multiplicar el valor de δ para la unidad de salida de la neurona por el valor z de la unidad de entrada de la neurona.

Veamos cómo calcular el gradiente local δ_k considerando que la neurona k puede encontrarse en la capa de salida o en una capa oculta. El primer caso se reduce a calcular el error e_k aplicando la ecuación (15.20). Sin embargo, si la neurona k se encuentra en la capa oculta no hay una forma directa de calcular el error, ya que no se conoce la salida deseada en ese punto. Así, este error tendrá que determinarse recursivamente considerando todas las neuronas de la capa de salida a las que está conectada dicha neurona k . Retomando la notación previa (véase la Figura 15.3), se denomina j a la neurona de la capa oculta de la que se desea estimar el gradiente, y k a cada una de las neuronas de salida. El gradiente local δ_j para una neurona en la capa oculta se define como:

$$\delta_j = -\frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial a_j} = -\frac{\partial E}{\partial z_j} g'(a_j). \quad (15.25)$$

De nuevo se emplea la regla de la cadena para calcular la primera derivada parcial:

$$\frac{\partial E}{\partial z_j} = \sum_k e_k \frac{\partial e_k}{\partial z_j} = \sum_k e_k \frac{\partial e_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} = \sum_k e_k \tilde{g}'(a_k) w_{kj} = -\sum_k \delta_k w_{kj}, \quad (15.26)$$

por tanto, retomando la ecuación (15.25):

$$\delta_j = g'(a_j) \sum_k \delta_k w_{kj} \quad (15.27)$$

y el ajuste del peso w_{ji} , de modo similar a la ecuación (15.24), viene dado por:

$$\Delta w_{ji} = \eta \delta_j x_i. \quad (15.28)$$

Es importante recordar que el error que se ha calculado es el error cometido al introducir el patrón n -ésimo, es decir, es un error individualizado. Por ello, los pesos se actualizan cada vez que se le presenta un nuevo patrón \mathbf{x}^n a la red. Este modo de aprendizaje se conoce como aprendizaje estocástico. Otra opción existente consiste en calcular las derivadas una vez que se han proporcionado a la red todos los patrones (aprendizaje por lotes o *batch*), de este modo los pesos de la primera capa se actualizarían como:

$$\Delta w_{ji} = -\eta \sum_n \delta_j^n x_i^n,$$

con expresiones análogas para los pesos de la segunda capa. Una de las ventajas de este algoritmo es su eficiencia computacional, ya que está demostrado que el coste computacional es $\mathcal{O}(W)$, donde W es el número total de pesos y umbrales.

15.2.2.2 Determinación de la topología del Perceptrón Multicapa

En 1989, Cybenko [Cybenko, 1989] demostró que el MLP con una sola capa oculta y un número suficiente de neuronas en dicha capa, con funciones de transferencia \tilde{g} sigmoideas, y una función de transferencia en la capa de salida lineal es capaz de aproximar cualquier función continua f de R^D en R con el grado de precisión deseado. Asimismo, Cybenko [Cybenko, 1989] también demostró que dicho perceptrón es un clasificador universal. Por lo tanto, cuando se aplica el perceptrón multicapa a un problema real, y no se consigue la precisión deseada, es porque no se ha conseguido una determinación adecuada de los pesos de la red, o no se ha utilizado el número apropiado de neuronas en la capa oculta. Veamos con un ejemplo el efecto que tiene sobre la capacidad de generalización de una red una selección inadecuada del número de neuronas de esta capa.

Supongamos un problema de regresión en el que la función ideal que debería modelar el MLP es la dada por la ecuación $y = \cos(0,7x) + \sin(0,3x)$. Como es habitual en la práctica real, los datos utilizados en el entrenamiento vienen acompañados de ruido. De este modo, se generó un conjunto de entrenamiento tomando 41 datos de esta distribución y añadiendo ruido gaussiano con desviación estándar (0,1). La Figura 15.5(a) presenta este ejemplo. La línea discontinua muestra la distribución ideal que debería aproximar la red y las cruces indican los 41 puntos que componen el conjunto de entrenamiento. La Figura 15.5(b) muestra las salidas proporcionadas por un MLP con una buena aproximación del modelo a estimar. Dicho MLP consta de 6 neuronas en su única capa oculta, con funciones de transferencia sigmoideas y una neurona de salida con función de transferencia lineal. Por otra parte, la Figura 15.6 presenta dos situaciones no deseables. La primera de ellas (Figura 15.6(a)), se debe al reducido número de neuronas usado en la capa oculta, lo que provoca que el MLP no sea capaz de estimar la función subyacente a los datos. Por el contrario, un excesivo número de neuronas en la capa oculta conduce a la segunda situación, conocida como *sobreajuste*. En esta situación, mostrada en la Figura 15.6(b), la red realiza una interpolación

exacta o, lo que es lo mismo, modela no sólo la función elegida sino también el ruido. No hay un procedimiento sencillo ni general para determinar la topología adecuada del MLP. Normalmente, se comienza por redes de tamaño reducido, dado que su entrenamiento es más rápido, y se va incrementando paulatinamente el número de neuronas hasta alcanzar una topología que presente el comportamiento deseado.

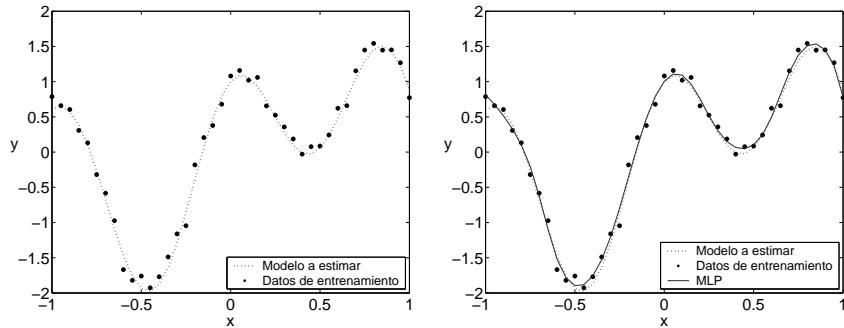


Figura 15.5: Ejemplo de regresión usando un MLP.

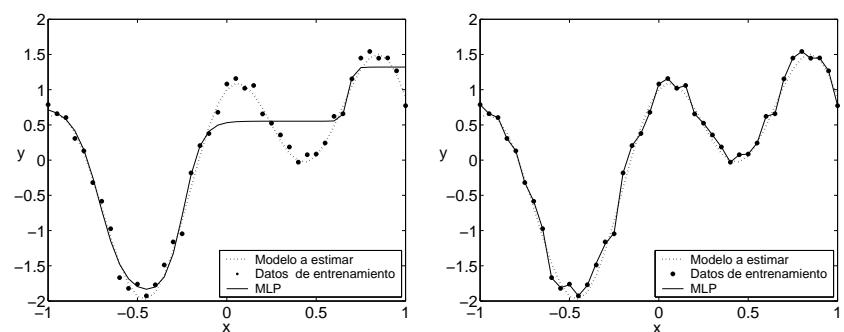


Figura 15.6: Efecto del número de neuronas ocultas sobre la capacidad de ajuste de un MLP.

15.2.2.3 Métodos avanzados de aprendizaje

En las secciones anteriores hemos visto el algoritmo básico de retropropagación del error para el entrenamiento de MLP. Este algoritmo utiliza la información del gradiente proporcionado por la derivada de una función de error respecto a los pesos para encontrar los valores de éstos que proporcionan un mínimo de la función de error. A pesar de su utilidad, este método presenta varios problemas:

1. La función de error a optimizar normalmente presenta varios mínimos locales (véase Figura 15.4). Esto implica que, cuando se finaliza el entrenamiento, no es posible saber si los valores obtenidos para los pesos se corresponden con un mínimo local o global, ni tampoco a qué distancia nos hallamos del mínimo global. De hecho, simplemente con utilizar distintos valores iniciales para los pesos podremos obtener soluciones finales diferentes tras el entrenamiento.
2. La convergencia del proceso de aprendizaje hacia el mínimo puede ser lenta.

Para resolver estos problemas han surgido otros métodos de aprendizaje, la mayoría basados en modificaciones del algoritmo original de retropropagación. Algunos de los más utilizados son:

- *Métodos basados en mínimos cuadrados lineales.* Se basan igualmente en la minimización del ECM pero lo hacen entre la señal a que recibe la neurona de salida antes de aplicar la función de activación no lineal y una versión de la salida deseada modificada adecuadamente. Con ello se consigue eliminar en algunos casos, y reducir en otros, las iteraciones necesarias en el entrenamiento [Castillo y otros, 2002, 2006; Cherkassky y Mulier, 1998; Fontenla-Romero y otros, 2003; Yam y otros, 1997].
- *Métodos de segundo orden.* El algoritmo de retropropagación utiliza la información proporcionada por la primera derivada de la función de error. Sin embargo, se ha demostrado que el uso de las derivadas segundas incrementa la velocidad del aprendizaje. Los algoritmos que las usan, llamados algoritmos de segundo orden, están entre los más rápidos en términos de convergencia. Algunos de los más utilizados son el algoritmo *BFGS Quasi-Newton* [Dennis y Schnabel, 1983], el *Levenberg-Marquardt* [Hagan y Menhaj, 1994] y el de *Gradiente Conjugado* [Moller, 1993; Powell, 1977]. No obstante, estos métodos implican un coste computacional considerable, por lo que no siempre resultan practicables cuando se manejan RNA y/o conjuntos de entrenamiento de gran tamaño [LeCun y otros, 1998].
- *Paso de aprendizaje adaptativo.* En el algoritmo de retropropagación la tasa de aprendizaje es constante; sin embargo, variar su valor podría acelerar y mejorar el aprendizaje. Se han desarrollado diversos métodos heurísticos para la adaptación dinámica de la tasa de aprendizaje [Hush y Salas, 1988; Jacobs, 1988; Vogl y otros, 1988], así como para la autodeterminación de este parámetro [Almeida y otros, 1999; Orr y Leen, 1996; Schraudolph, 2002; Weir, 1991].

- *Inicialización apropiada de los pesos.* El punto de partida del entrenamiento, determinado por los valores iniciales de los pesos, también influye en la calidad de la convergencia hacia el mínimo de la función de error. Por este motivo, se han propuesto diversos esquemas para la correcta inicialización de los pesos. Quizás uno de los más conocidos es el propuesto por Nguyen y Widrow [Nguyen y Widrow, 1990] que asigna a cada neurona oculta una porción del rango de la respuesta deseada.
- *Versiones por lotes y estocástica.* Como se ha visto, el algoritmo de retropropagación del error permite calcular el error y actualizar los pesos, bien sobre todo el conjunto de entrenamiento, bien ejemplo a ejemplo. Esta opción puede acelerar el tiempo de convergencia cuando se manejan conjuntos de entrenamiento de gran tamaño en problemas de clasificación [LeCun y otros, 1998].

15.2.3 Redes de base radial

Los tipos de redes analizados en secciones anteriores se caracterizan por utilizar neuronas cuya salida es una función (normalmente no lineal) del producto escalar entre el vector de entradas y un vector de pesos. En esta sección se explican las *Redes de Neuronas con Funciones de Base Radial*², en adelante RBFN, en las que el nivel de activación de una neurona oculta estará determinado por la *distancia* entre el vector x de entrada a la red y un vector prototípico asociado a dicha neurona.

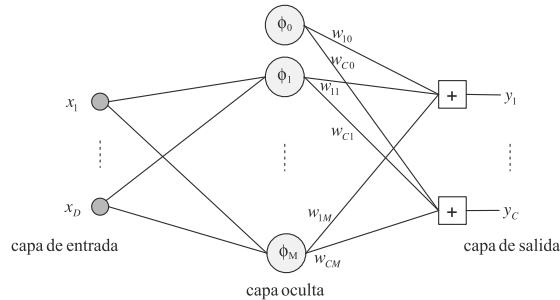


Figura 15.7: Arquitectura general de una red de base radial.

15.2.3.1 Arquitectura de las redes de base radial

Tal como se puede observar en la Figura 15.7, las redes de base radial presentan una arquitectura muy sencilla compuesta únicamente por dos capas. Una primera y única capa oculta es la encargada de preprocesar los patrones de entrada. Recibe los vectores de entrada $\mathbf{x} = \{x_1, x_2, \dots, x_D\}$ y, a través de las funciones base ϕ_j que

²En inglés, *Radial Basis Function Networks*

contiene, realiza una transformación no lineal del espacio de entradas D -dimensional hacia otro espacio, normalmente de dimensión menor.

Existen distintas formas para las funciones base no lineales $\phi(\cdot)$, y en general, se ha demostrado [Bishop, 1995] que las propiedades finales de la red son relativamente insensibles a la forma de estas funciones. Cualquier función no lineal basada en una medida de distancia es válida, especialmente si cumple dos propiedades: ser *localizada* y *radialmente simétrica* (de ahí el nombre de estas redes). La primera propiedad implica que la función sólo se activará en una región delimitada del espacio de entradas, es decir, $\phi(\mathbf{x}) \rightarrow 0$ cuando $|\mathbf{x}| \rightarrow \infty$. La segunda propiedad implica funciones con un único máximo en el origen que decaen simétricamente a ambos lados rápidamente hacia cero. La función *gaussiana* cumple todas estas características, y es por ello la función de base radial más empleada. Presenta la forma de la Figura 15.8 y está determinada por la ecuación:

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right), \quad (15.29)$$

donde $\boldsymbol{\mu}_j$ es el vector que determina el centro de la función base ϕ_j , y σ_j es un parámetro cuyo valor controla la pendiente con que la función se aleja de su centro, como se observa en la Figura citada. Es importante recalcar aquí que cada neurona oculta j tendrá sus propios parámetros σ_j y $\boldsymbol{\mu}_j$, que determinarán la forma concreta de su función base y que se establecerán durante el aprendizaje.

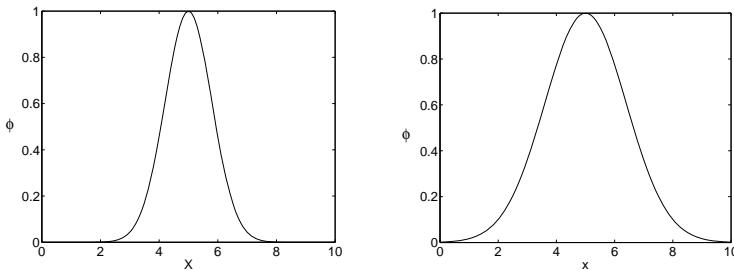


Figura 15.8: Funciones gaussianas con valores $(\mu = 5, \sigma = 0,8)$ y $(\mu = 5, \sigma = 1,4)$, respectivamente.

La segunda capa de una RBFN se corresponde con un perceptrón simple (véase Figura 15.7). Pondera la salida de las neuronas de la capa oculta utilizando un vector de pesos \mathbf{w} de elementos w_{kj} para, finalmente, producir las salidas de la red:

$$y_k = \sum_{j=0}^M w_{kj} \phi_j(\mathbf{x}); \quad k = 1, 2, \dots, C. \quad (15.30)$$

donde, igual que en secciones anteriores, los pesos w_{k0} representan los sesgos, que se introducen en la suma general incluyendo una función base constante adicional $\phi_0 = 1$.

Aunque en este caso las neuronas de salida tan sólo realizan una suma, podrían incluir funciones de activación lineales o sigmoideas. También es posible crear topologías más generales y complejas para este tipo de redes, pero la explicada y mostrada en la Figura 15.7 es la arquitectura que normalmente se utiliza. Como se verá más adelante, el número M de funciones base necesarias, o lo que es lo mismo el tamaño de la capa oculta, dependerá de la complejidad del problema a representar, sin que en ello influya el tamaño del conjunto de entrenamiento.

15.2.3.2 Comportamiento de las redes de base radial

Existen diversos estudios [Hartman y otros, 1990; Park y Sandberg, 1991] que demuestran que la superposición lineal de funciones localizadas (como es el caso de la RBFN definida por las ecuaciones (15.30) y (15.29)) es capaz de aproximar cualquier tipo de función. Aunque estos estudios no proporcionan el modo de obtener esa red ideal y suponen que es posible poder construir RBFN con un gran número de funciones base, sus resultados teóricos siguen siendo de gran interés práctico.

Para ilustrar el funcionamiento de las RBFN podemos considerar el problema de regresión descrito ya en la sección 15.2.2.2 en el que, a partir de unos datos de entrenamiento, se trataba de encontrar el modelo que los generaba definido por la ecuación $y = \cos(0,7 * x) + \sin(0,3 * x)$. Inicialmente, se entrenó una RBFN con tantas neuronas ocultas como datos de entrenamiento (es decir, 41) y una sola neurona de salida, y se fijó $\sigma_j = 0,98, \forall j$. Como se puede observar en la Figura 15.9(a), tal como ocurría con el perceptrón, un exceso de unidades ocultas hace que la red se sobreajuste, de manera que modela no sólo la función subyacente a los datos sino también el ruido.

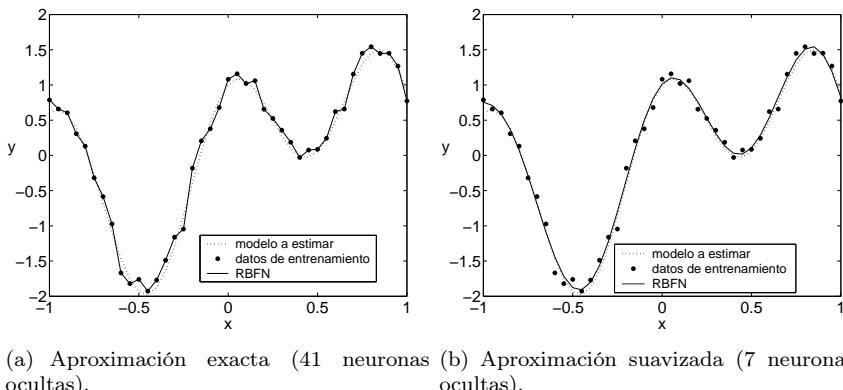


Figura 15.9: Efecto del número de neuronas ocultas sobre la capacidad de ajuste de una RBFN.

Como contraste, en la Figura 15.9(b) podemos ver el ajuste realizado por una RBFN en la que el número de funciones base es menor que el número N de datos de entrenamiento disponibles. En este caso, a medida que el número de funciones base disminuye respecto a N , también se suaviza la función de ajuste que genera la red, perdiendo la capacidad de un ajuste exacto a los datos y ganando en capacidad de generalización. Además de aumentar el tiempo de entrenamiento, un gran número de funciones base implican la necesidad de un mayor número de patrones de entrenamiento para poder ajustar los parámetros convenientemente y evitar el sobreajuste. En la práctica, es necesario conseguir un equilibrio entre el número de parámetros ajustables y la flexibilidad de la red.

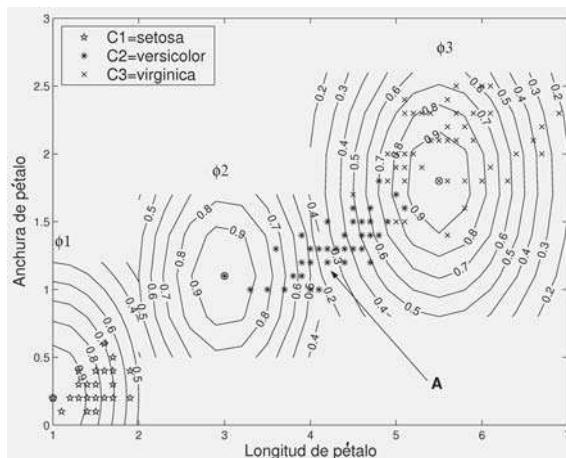


Figura 15.10: Regiones de separación, y los niveles de activación correspondientes, creadas para el problema de Iris por cada una de las 3 neuronas ocultas de una RBFN.

En cuanto a los problemas de clasificación, las RBFN dividen el espacio de entrada en subespacios, cada uno representado por un grupo reducido de neuronas ocultas que se activan cuando reciben un patrón de entrada perteneciente a dicho subespacio. La segunda capa combina la salida de las neuronas activas para proporcionar la respuesta final. Así, se construyen regiones cerradas de decisión entre grupos del espacio de entrada. En la Figura 15.10 se pueden observar las regiones de decisión que una RBFN, con 3 neuronas ocultas y otras tantas de salida, genera para el conocido problema de clasificación de plantas de Iris³. Las curvas concéntricas de contornos

³Se trata de un famoso conjunto de datos creado por Fisher para ilustrar los principios del análisis discriminante. Consta de 4 variables de entrada (anchura y longitud de pétalos y sépalos), aunque en este ejemplo se utiliza una versión simplificada que tiene en cuenta sólo las características de los pétalos. El objetivo es clasificar la planta de Iris en una de tres especies posibles (Setosa, Versicolor y Virginica). Los datos y su descripción pueden encontrarse en <ftp://ftp.ics.uci.edu/pub/machine-learning-databases/>

indican el nivel de activación que presentan, respecto al espacio de entradas, cada una de las funciones base de las 3 neuronas ocultas. Como se puede observar, estas neuronas dividen el espacio de entradas en 3 grupos.

Además, en este tipo de problemas, las salidas de las funciones base ϕ_j se pueden interpretar como la probabilidad $p(\text{grupo}_j \mid \mathbf{x}^n)$ de que un vector de entrada \mathbf{x}^n pertenezca al grupo de datos j abarcado por la función radial ϕ_j , mientras que los pesos w_{kj} de la segunda capa se interpretan como la probabilidad $p(C_i \mid \text{grupo}_j)$ de la clase C_i dada la presencia del grupo j [Bishop, 1995]. Si volvemos al ejemplo de la Figura 15.10, en presencia del patrón etiquetado como **A** que pertenece a la clase $C2=versicolor$, los niveles de activación de las funciones base ϕ_1 , ϕ_2 y ϕ_3 son, respectivamente, $0,4 \times 10^{-3}$, 0,37 y 0,24. Finalmente, tras combinar linealmente estos valores en la capa de salida la RBFN devuelve los valores 0,01 en la neurona asociada a la clase setosa, 1,0 en la asociada a la clase versicolor, y 0,09 en la asociada a la clase virginica, indicando que la mayor probabilidad es para la clase *versicolor*, y así clasificando correctamente el ejemplo.

15.2.3.3 El aprendizaje en las redes de base radial

Durante el entrenamiento de las RBFN, la idea es forzar a cada elemento oculto de la red para que represente una porción del espacio de entrada, de manera que cada uno de ellos contenga un *prototipo* de un grupo de ejemplos del conjunto de entrenamiento. Cuando se presente un nuevo ejemplo a la red se activará la neurona cuyo prototipo sea el más parecido al ejemplo y, a partir de esta activación, se determinará la salida de la red. Por lo tanto, hay dos conceptos que deben incluir las funciones de activación de las neuronas ocultas: el de prototipo, y el de *similitud* o *distancia* entre el prototipo y el ejemplo de entrada.

Para conseguir este objetivo, los parámetros de las funciones base de una RBFN se deben escoger de forma que representen la *densidad de probabilidad* de los datos de entrada. Gracias a esta interpretación estadística que se puede dar a las representaciones internas de las neuronas ocultas, el proceso de entrenamiento de las redes de base radial se divide en dos etapas. En la primera de ellas se establecen los parámetros que determinan las funciones base de las neuronas ocultas (i.e., μ_j y σ_j en el caso de funciones gaussianas). Esta etapa se realiza de modo no supervisado, es decir, teniendo en cuenta exclusivamente los vectores \mathbf{x} de entrada del conjunto de entrenamiento y no las salidas deseadas \mathbf{t} , y trata de situar los elementos ocultos de la red de manera que cubran aquellas regiones del espacio de entrada en las que hay datos. En una segunda etapa de entrenamiento los parámetros de las funciones base quedan fijos, y se determinan los pesos \mathbf{w} de la última capa. Será esta segunda etapa la que establezca la asociación entre las regiones formadas en el espacio de entrada y los datos de salida deseados para la red.

Una consecuencia de este tipo de entrenamiento es su rapidez ya que existen distintos métodos, todos ellos muy rápidos, para determinar los parámetros de la primera etapa, mientras que la segunda se reduce a resolver un problema lineal que resulta ser, por lo tanto, también muy rápido.

Primera etapa del entrenamiento. Como ya se ha comentado, en esta etapa, que se realiza de forma no supervisada, se trata de obtener los parámetros μ_j y σ_j que determinan la forma concreta de cada función base. No se debe olvidar que el objetivo es conseguir que las funciones base aproximen la distribución del espacio de entradas \mathbf{x} y que lo harán de forma local, es decir, cada neurona oculta representará una zona diferente en dicho espacio. Los centros μ_j se consideran *prototipos* de estos vectores de entrada, de modo que sólo se activarán las neuronas cuyos μ_j estén cerca del vector de entrada \mathbf{x}^n . El parámetro σ_j , que define la anchura de las funciones base, también influye en la calidad del ajuste realizado por la red una vez entrenada. En el ejemplo de regresión utilizado en la Figura 15.9(b) los parámetros σ_j de las 7 neuronas ocultas se fijaron al valor 0,57, es decir, aproximadamente el doble del espacio medio que hay entre los centros de las funciones. Sobre este mismo ejemplo, la Figura 15.11 muestra las consecuencias de utilizar valores de σ demasiados pequeños o demasiado grandes.

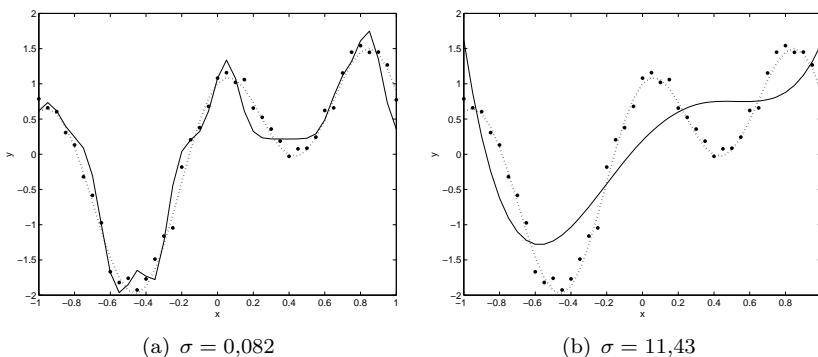


Figura 15.11: Efecto del parámetro σ en la RBFN. En el primer caso, se obtiene una función de salida con cambios excesivamente bruscos. En el segundo caso, una función excesivamente suavizada.

El rendimiento de una RBFN también depende de los centros escogidos para las funciones base. Existen distintos procedimientos para la optimización de estos valores, por ejemplo, igualar los centros μ_j a los valores de algunos de los vectores de entrenamiento, o utilizar algoritmos de agrupamiento. Uno de los métodos más relevantes es el basado en *modelos de mezcla gaussianos* que considera el problema como uno de estimación de la densidad del espacio \mathbf{x} a través de un modelo de mezclas cuyos componentes son las funciones base. Así, el modelo de densidad buscado se define por la mezcla:

$$p(\mathbf{x}) = \sum_{j=1}^M P(j) \phi_j(\mathbf{x}) \quad (15.31)$$

siendo $P(j)$ los coeficientes de mezclado que pueden asimilarse a la probabilidad a priori de los datos que genera la componente j de la mezcla. Los parámetros de este

modelo ($P(j)$, μ_j y σ_j) se pueden optimizar por máxima verosimilitud. Siendo la función de verosimilitud:

$$\ell = \prod_{n=1}^N p(\mathbf{x}^n) \quad (15.32)$$

ésta puede maximizarse calculando sus derivadas respecto a los parámetros de las funciones base y, posteriormente, utilizar estas derivadas en algoritmos de optimización no lineales estándares basados en el descenso de gradiente. Estos parámetros también pueden obtenerse mediante procedimientos de reestimación basados en el algoritmo EM (del inglés, *Expectation-Maximization*) [Dempster y otros, 1977]. En cualquier caso, una vez optimizada la mezcla expresada en la ecuación 15.31, los coeficientes $P(j)$ pueden desecharse, mientras que los coeficientes de las funciones ϕ_j serán los que utilicen las funciones base de la red.

Otro algoritmo utilizado en esta primera etapa es el de *mínimos cuadrados ortogonales* [Chen y otros, 1991]. Este método consiste en introducir y ajustar las unidades ocultas de forma secuencial. En cada paso del algoritmo se introduce una neurona oculta y se fija su prototipo μ_j al valor del patrón \mathbf{x}^n del conjunto de aprendizaje que maximiza el incremento de la varianza de la salida deseada explicada por la red o, lo que es lo mismo, minimiza el error que se obtendría sobre el conjunto de entrenamiento. Además, es posible la selección directa del dato que debe ser elegido como prototipo de la siguiente neurona, gracias a la construcción de una base de vectores ortogonales en el espacio abarcado por las activaciones de las neuronas ocultas. En cualquier caso, para obtener una buena capacidad de generalización, el número de neuronas ocultas que se obtengan deberá ser menor que el tamaño del conjunto de entrenamiento.

Segunda etapa del entrenamiento. Como ya se ha mencionado, en esta etapa los parámetros de las funciones base permanecen fijos y se trata de determinar los pesos \mathbf{w} de la última capa que minimizan un error respecto a la función a aproximar. Supóngase que la función de error utilizada es el error cuadrático medio, ya explicada y definida por:

$$E = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^C \{y_k^n - t_k^n\}^2 \quad (15.33)$$

Teniendo en cuenta que la salida y_k se calcula de acuerdo a la ecuación (15.30) que es una función lineal de los pesos, la función de error será cuadrática en los pesos:

$$E = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^C \left\{ \sum_{j=0}^M w_{kj} \phi_j(\mathbf{x}^n) - t_k^n \right\}^2 \quad (15.34)$$

De este modo, al derivar esta función respecto a los pesos w_{kj} e igualarla a cero para obtener su mínimo, vemos que el conjunto de pesos que minimiza el error sobre el conjunto de entrenamiento puede calcularse de forma exacta mediante la solución del siguiente sistema de ecuaciones lineales:

$$\sum_{n=1}^N \left\{ \sum_{j'=0}^M w_{kj'} \phi_{j'}(\mathbf{x}^n) - t_k^n \right\} \phi_j(\mathbf{x}^n) = 0; \quad k = 1, 2, \dots, C \text{ y } j = 1, 2, \dots, M. \quad (15.35)$$

En efecto, se considera la notación matricial de la ecuación (15.35), los pesos están determinados por las ecuaciones lineales

$$\Phi^T \Phi \mathbf{W}^T = \Phi^T \mathbf{T} \quad (15.36)$$

donde \mathbf{W} , Φ y \mathbf{T} son matrices de componentes w_{kj} , $\phi_j(\mathbf{x}^n)$ y t_k^n , respectivamente. Finalmente, la solución para los pesos viene dada por

$$\mathbf{W}^T = \Phi^\dagger \mathbf{T} \quad (15.37)$$

siendo Φ^\dagger la matriz seudoinversa de Φ . De este modo, los pesos de la segunda capa pueden encontrarse aplicando técnicas rápidas y lineales de inversión de matrices [Bishop, 1995].

15.2.3.4 Comparación con el perceptrón multicapa

Las redes de base radial y los perceptrones multicapa son técnicas que comparten el hecho de permitir la aproximación de funciones no lineales en espacios multidimensionales. En ambos casos, esta aproximación se consigue mediante la composición parametrizada de funciones básicas de una variable, y los parámetros se determinan tras un proceso a partir de un conjunto de datos representativos del problema a resolver. Sin embargo, la diferente estructura que presenta cada tipo de red determina ciertas diferencias entre ellas que conviene conocer:

1. Respecto a la capacidad para aproximar funciones, Girosi y Poggio [Girosi y Poggio, 1990] demostraron que las RBFN presentan la propiedad de *mejor aproximación* que establece que, entre el conjunto de todas las funciones que se pueden construir según los parámetros ajustables, existe una que comete el mínimo error, sea cual sea la función a aproximar. Esta propiedad no se cumple en el caso de los MLP.
2. En general, el proceso de entrenamiento de las redes de base radial pueden ser significativamente más rápido que los métodos empleados para el entrenamiento de los MLP. Una RBFN realiza una representación localizada de los datos de entrada, es decir, para un patrón de entrada determinado sólo unas pocas neuronas ocultas se activarán de forma significativa. Además, las dos capas se entrena de forma independiente y, para ambas, es posible utilizar técnicas rápidas. Por contra, los MLP realizan una representación distribuida, y son muchas las neuronas ocultas que contribuirán a determinar el valor de salida de la red. Las interferencias entre estas neuronas complican el proceso de entrenamiento que resulta altamente no lineal, por lo que se resuelve mediante técnicas de descenso de gradiente, con problemas conocidos como el de los mínimos locales. Como consecuencia, es posible que los tiempos de entrenamiento sean largos, incluso para métodos sofisticados.
3. Por las razones expuestas en el punto anterior, los resultados de una RBFN son más fáciles de interpretar.

4. Debido a la representación local de sus neuronas ocultas, generalmente las RBFN necesitan más neuronas ocultas y ejemplos de entrenamiento que los MLP.
5. Normalmente, es más difícil encontrar conjuntos de datos etiquetados que no etiquetados (es decir, datos x que carecen del correspondiente t), ya que se requiere del tiempo y del conocimiento de un experto humano para hacerlo. Las RBFN presentan la ventaja de que su primera etapa de entrenamiento es no supervisada e independiente de la segunda. Por lo tanto, es posible utilizar todos los datos no etiquetados en esta primera etapa más compleja, y emplear el conjunto más reducido de datos etiquetados sólo en la segunda fase del aprendizaje.
6. Una desventaja de las RBFN es su incapacidad para distinguir variables de entrada que no son significativas para resolver el problema. El problema se agrava si estas variables presentan una varianza significativa en el conjunto de aprendizaje. Esto sucede, porque los centros de las funciones base se escogen para representar exclusivamente los datos de entrada. En este aspecto, se ha demostrado [Bishop, 1995] el comportamiento mejor del MLP.
7. Por último, las regiones de clasificación cerradas que construyen las RBFN las hacen inapropiadas para aprender transformaciones lógicas tales como funciones booleanas o de paridad [Moody y Darken, 1989].

15.3 Métodos de aprendizaje no supervisados

Las redes de neuronas comentadas en los apartados anteriores son sistemas basados en el aprendizaje supervisado. Sin embargo, existen también otros paradigmas neuronales basados en un aprendizaje de tipo no supervisado o no guiado. La idea común de todos ellos es tratar de representar los datos de entrada del sistema de forma que esta representación refleje la estructura estadística que los define. En otras palabras, tratar de encontrar patrones o características que sean significativas en los datos de entrada. En este caso no se dispone de ninguna salida con la que comparar el rendimiento del método, por lo tanto, la única información que tendrá el sistema es la que proporcionen los datos de entrada.

Este tipo de sistemas se pueden clasificar en tres grandes grupos, dependiendo del esquema de organización interna que empleen para actualizar las características de las neuronas de la red:

- Métodos de aprendizaje basados en la regla de Hebb. Una de las primeras aproximaciones al aprendizaje no supervisado fue la realizada por Hebb (véase sección 15.2.1.1) y que ha supuesto una de las aportaciones fundamentales en el campo de las redes de neuronas. Basados en ella, se han propuesto diversos tipos de reglas *hebbianas* empleadas con diversos fines, aunque una de las aplicaciones más importantes es en el *análisis de componentes principales*.
- Métodos de aprendizaje competitivos. En este tipo de sistemas el objetivo es agrupar o categorizar los datos que sean similares. Generalmente, se trata de

similitud geométrica en el espacio de entrada. En estos modelos existe un conjunto de neuronas que compiten para ser activadas. A diferencia de los métodos anteriores, donde varias neuronas pueden ser activadas al mismo tiempo, en este tipo de sistemas sólo una de ellas será la *ganadora*. Por tanto, la red tratará de agrupar los patrones similares que serán representados por una neurona de la red. Uno de los algoritmos más empleados es el de los *mapas autoorganizativos* que será tratado con algo más de profundidad en las secciones siguientes.

- Métodos de aprendizaje basados en modelos de teoría de la información. En 1988 Linsker [Linsker, 1988] propuso por primera vez un esquema de aprendizaje basado en los principios de la *teoría de la información*. Este tipo de sistemas trata de maximizar la cantidad de información que se conserva en el procesamiento de los datos, o lo que es lo mismo, minimizar la *entropía*. Este tipo de métodos tiene multitud de aplicaciones prácticas, entre ellas, el análisis de componentes principales y sistemas de compresión de información.

15.3.1 Mapas autoorganizativos

El mapa autoorganizativo propuesto por Kohonen [Kohonen, 1995] es un sistema usado para visualizar e interpretar conjuntos de datos con un espacio de entrada de alta dimensionalidad. Este sistema convierte las relaciones estadísticas complejas entre los datos de entrada en relaciones geométricas simples en otro espacio de baja dimensión.

El mapa autorganizativo está formado por una malla de P neuronas, habitualmente bidimensional, como la mostrada en la Figura 15.12(a). En este ejemplo, la malla está compuesta por un conjunto de 10×10 elementos. La Figura 15.12(b) muestra el estado de la malla después del aprendizaje sobre un determinado conjunto de datos. Cada neurona de la malla está definida por un vector de referencia \mathbf{w}_j , $j = 1, \dots, P$, que toma valores en el mismo espacio que los datos de entrada.

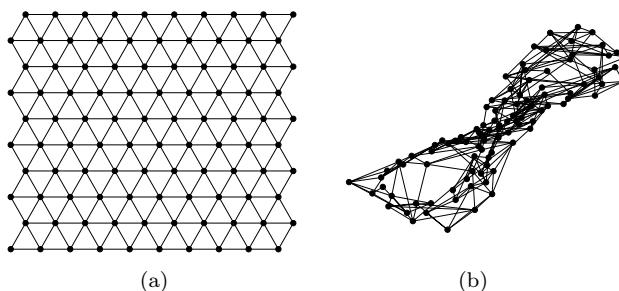


Figura 15.12: Ejemplos de mallas de neuronas de un mapa autoorganizativo, (a) antes del entrenamiento y (b) después del entrenamiento.

Una de las diferencias principales del mapa autoorganizativo frente a otros sistemas de aprendizaje competitivos, es que en cada paso del aprendizaje se actualizan no sólo los pesos de la neurona ganadora, sino también los pesos del resto de neuronas de la malla. En primer lugar, dado un vector de entrada \mathbf{x}^n , $n = 1, \dots, N$ el sistema calcula cuál es la neurona *ganadora* como aquella que más se parece a la entrada, es decir, la que presenta una menor distancia a ese dato:

$$c = \arg \min_j d(\mathbf{w}_j - \mathbf{x}^n), \quad j = 1, \dots, P, \quad (15.38)$$

donde c es el índice de la neurona ganadora y la función $d(\cdot)$ es una distancia métrica arbitraria. Si, por ejemplo, se emplea la distancia euclídea el término $d(\mathbf{w}_j - \mathbf{x}^n)$ se transforma en $\|\mathbf{w}_j - \mathbf{x}^n\|$.

Posteriormente, se lleva a cabo la actualización del sistema. En el mapa autoorganizativo la actualización de los pesos en el paso $\tau + 1$ del algoritmo se realiza aplicando la siguiente regla:

$$\mathbf{w}(\tau + 1) = \mathbf{w}(\tau) + \lambda(\tau) h_{c,j}(\tau) [\mathbf{x}^n - \mathbf{w}(\tau)] \quad (15.39)$$

donde $h_{c,j}(\tau)$ es una función de vecindad espacial entre neuronas y $\lambda(\tau)$ es el paso de aprendizaje en el instante τ . Una de las funciones de vecindad empleadas habitualmente es la función de tipo gaussiana definida por la siguiente ecuación:

$$h_{c,j}(\tau) = \exp \left(-\frac{\|\mathbf{r}_c - \mathbf{r}_j\|^2}{2\sigma^2(\tau)} \right) \quad (15.40)$$

donde $\sigma(\tau)$ es un parámetro que indica la anchura de la función gaussiana y $\mathbf{r}_c \in \Re^2$ y $\mathbf{r}_j \in \Re^2$ son los vectores de posición de las neuronas c y j dentro de la malla. Si el valor de $\|\mathbf{r}_c - \mathbf{r}_j\|$ es grande (i.e., existe una gran distancia entre las neuronas c y j) entonces la función $h_{c,j}$ tiende a cero, por tanto los pesos de las neuronas de la malla lejanas a la neurona ganadora, c , se incrementan en menor medida en la ecuación (15.39).

Como la gran mayoría de los sistemas neuronales, el mapa autoorganizativo tiene dos modos de funcionamiento:

1. Durante el proceso de aprendizaje, empleando los datos de entrenamiento del sistema, se adapta el mapa mediante un proceso competitivo entre las neuronas.
2. Una vez entrenado, durante el proceso de operación, el sistema recibe un nuevo dato que es asociado a la neurona ganadora del mapa. Esta neurona es la que presenta la menor distancia respecto del dato recibido. La salida que produzca esta neurona será la salida de la red para ese dato.

15.4 Máquina de Vectores Soporte

Los fundamentos de las Máquinas de Vectores Soporte (*Support Vector Machines*, SVM) fueron desarrollados por Vapnik [Vapnik, 1995, 1998], en unos trabajos sobre

las teorías de aprendizaje estadístico que pretendían acotar el error de generalización en función de la complejidad del espacio de búsqueda. Las máquinas de vectores soporte están fundamentadas en sólidos principios teóricos y proporcionan un buen rendimiento en gran variedad de aplicaciones prácticas [Cristianini y Shawe-Taylor, 2000; Smola y Schölkopf, 2002]. Su formulación abarca el principio de minimización del riesgo estructural (SRM, *Structural Risk Minimization*), en lugar de la minimización del riesgo empírico (ERM, *Empirical Risk Minimization*) empleada en la mayoría de los métodos de aprendizaje estadístico. Es decir, las SVM no se centran en construir sistemas que cometan pocos errores, sino que intentan construir modelos fiables en los cuales se pueda tener una gran confianza, aunque se cometan algunos errores más en el entrenamiento del sistema. Es esta diferencia la que proporciona a las SVM una buena capacidad de generalización. Por tanto, las SVM tratan de obtener modelos que estructuralmente tengan poco riesgo de cometer errores ante datos futuros. Aunque en un principio se diseñaron para resolver problemas de clasificación binaria (dos clases), su aplicación se ha extendido a tareas de regresión, multiclasicación, agrupamiento, etcétera.

A modo de breve explicación, se tratará a continuación el problema de clasificación binaria más simple; aquél en que las dos clases son separables linealmente. Posteriormente se ampliará a la solución del caso no separable y de múltiples clases.

15.4.1 Caso linealmente separable

En el caso de un problema de clasificación binario, en el que las dos clases son separables mediante una recta (en general por un hiperplano), existen múltiples soluciones, puesto que pueden existir diversos hiperplanos que separan correctamente los ejemplos de ambas clases (sean capaces de minimizar el error empírico). La Figura 15.13(a) muestra un ejemplo de un problema de clasificación binaria de este tipo. Un ejemplo de las rectas posibles que discriminan correctamente entre ambas clases puede observarse en la Figura 15.13(b).

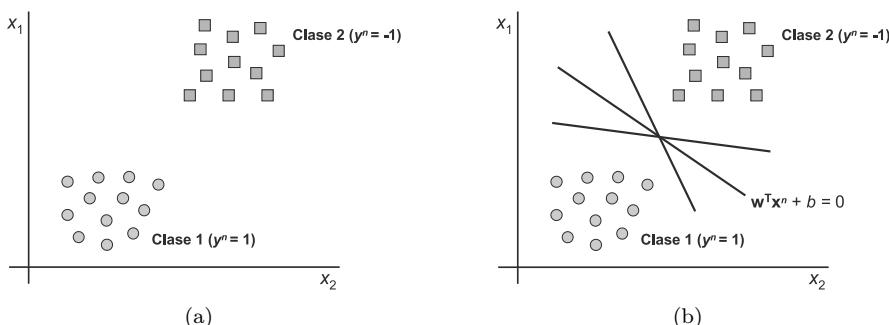


Figura 15.13: Ejemplo de clasificación lineal para dos clases.

Sin embargo, de manera intuitiva, parece claro que no todas las rectas que consiguen separar los ejemplos sin cometer errores son igualmente buenas en cuanto al error de generalización. Este hecho se puede apreciar visualmente en las Figuras 15.14(a) y 15.14(b). En cada una de estas figuras se representan dos de las posibles rectas que discriminan entre ambas clases. Además de dicha recta discriminatoria, se representan también las dos rectas paralelas (líneas discontinuas) a la anterior que pasan por los puntos de cada clase que están más cercanos a la recta de decisión. Estas rectas paralelas definen un margen entre dichos puntos. Intuitivamente se puede apreciar que la recta que presenta un margen mayor es la que producirá menos errores con nuevos datos, y por tanto cometerá un menor error de generalización. Este hecho fue demostrado teóricamente por Vapnik en sus trabajos, de los cuales se puede concluir que de todos los posibles hiperplanos que minimizan el error empírico (error sobre el conjunto de entrenamiento), se debe elegir aquel que presente un mayor margen respecto a los datos.

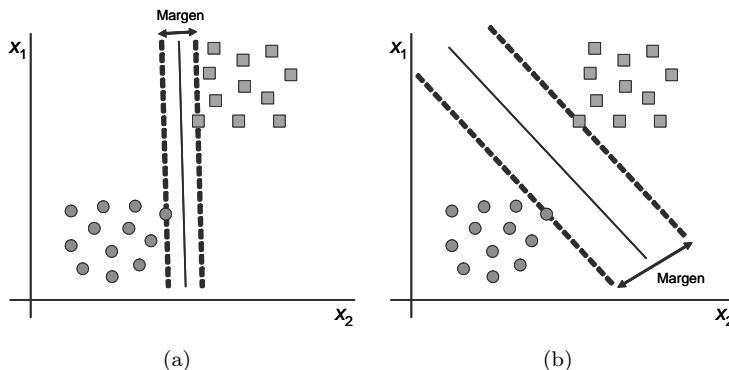


Figura 15.14: Rectas de decisión con diferente tamaño de margen.

A continuación, se tratará de definir formalmente este planteamiento para poder obtener la solución a este problema. Dado un conjunto de N datos, \mathbf{x}^n , $n = 1, \dots, N$, y sus correspondientes clases, $y^n \in \{+1, -1\}$, $n = 1, \dots, N$, se tratará de encontrar el hiperplano $\mathbf{w}^T \mathbf{x} + b = 0$ que sea capaz de discriminar entre ambas clases, y que presente el mayor margen respecto a los datos, donde \mathbf{w} representa la pendiente y b el desplazamiento al origen. Para ello, se definen dos hiperplanos, H_1 y H_2 , paralelos al hiperplano discriminante tal que todos los datos de una de las clases queden por debajo de H_1 y todos los de la otra queden por encima de H_2 :

$$\begin{aligned} H_1: \quad & \mathbf{w}^T \mathbf{x}^n + b \geq 1 && \text{para } y^n = 1 \\ H_2: \quad & \mathbf{w}^T \mathbf{x}^n + b \leq -1 && \text{para } y^n = -1 \end{aligned} \quad (15.41)$$

Puesto que la distancia perpendicular de H_1 y H_2 al origen es:

$$\begin{aligned} d_{H_1} &= \frac{|b - 1|}{\|\mathbf{w}\|} \\ d_{H_2} &= \frac{|b + 1|}{\|\mathbf{w}\|} \end{aligned} \quad (15.42)$$

Se obtiene que la distancia o *margen* entre ambos hiperplanos es:

$$\text{margen} = |d_{H_1} - d_{H_2}| = \frac{2}{\|\mathbf{w}\|} \quad (15.43)$$

La Figura 15.15 ilustra gráficamente el significado de los hiperplanos definidos y de las distancias calculadas para datos de entrada bidimensionales.

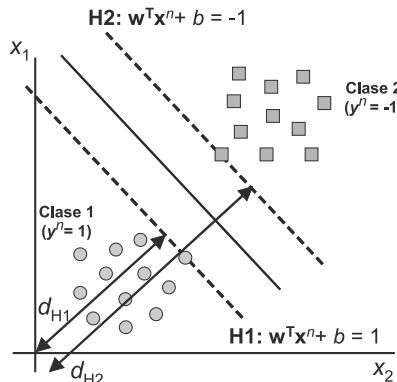


Figura 15.15: Ejemplo de los hiperplanos H_1 y H_2 y del margen asociado.

El objetivo será entonces identificar, entre todos los hiperplanos válidos, aquél que presente un mayor margen, puesto que presentará un mejor error de generalización. Por tanto, esta meta se plantea como un problema de minimización, ya que para maximizar el margen definido en la ecuación (15.43) habrá que minimizar el divisor, es decir, $\|\mathbf{w}\|$. Por consiguiente, el objetivo será obtener la solución del siguiente problema de optimización:

$$\begin{aligned} \text{Minimizar} \quad E(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sujeto a} \quad y^n (\mathbf{w}^T \mathbf{x}^n + b) - 1 &\geq 0; \quad \forall n = 1, \dots, N. \end{aligned} \quad (15.44)$$

Las restricciones incluidas son las presentadas en la ecuación (15.41) pero reescritas en una sola ecuación en función de la clase y^n correspondiente a cada dato. Por otro lado, se puede observar que en la función a minimizar se ha incluido la constante $1/2$ y el cuadrado de la norma. Esto no influye en la solución óptima pero

facilita los cálculos a la hora de resolver el problema ya que la función será derivada posteriormente. Resolviendo el problema planteado en la ecuación (15.44), mediante métodos de optimización de funciones se obtiene el hiperplano de margen geométrico máximo que clasifica correctamente todos los datos de entrenamiento. Sin embargo, el problema presentado suele replantearse empleando la formulación dual (*problema dual*). Esto tiene un doble objetivo: obtener un problema, por un lado, más fácil de resolver, y por otro lado, que permita generalizar más fácilmente el método al caso de datos que no sean linealmente separables.

Para obtener el problema dual se aplican los *multiplicadores de Lagrange* al problema planteado en la ecuación (15.44), dando lugar a la siguiente función de Lagrange del problema primario:

$$L(\mathbf{w}, b; \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \alpha_n [y^n (\mathbf{w}^T \mathbf{x}^n + b) - 1], \quad (15.45)$$

donde $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)$ es el vector de multiplicadores no negativos de Lagrange. A los multiplicadores de Lagrange también se les denomina *variables duales*, en contraposición a las variables del problema primal (i.e., \mathbf{w} y b). Intuitivamente, indican la importancia de cada restricción, de tal forma que, a mayor valor, más difícil es cumplir la restricción. A partir de la función lagrangiana se puede definir el *problema dual*. Este problema se obtiene buscando un punto estacionario de la función lagrangiana, el cual puede lograrse diferenciando dicha función respecto a las variables primarias e igualándolas a cero:

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b; \boldsymbol{\alpha}) = \mathbf{w} - \sum_{n=1}^N \alpha_n y^n \mathbf{x}^n = 0 \quad (15.46)$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b; \boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n y^n = 0 \quad (15.47)$$

A partir de la ecuación (15.46) se tiene que:

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y^n \mathbf{x}^n. \quad (15.48)$$

Sustituyendo, el resultado mostrado en la ecuación (15.48), en la función de Lagrange definida previamente en la ecuación (15.45) se obtiene:

$$L(\mathbf{w}, b; \boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{j=1}^N y^n y^j \alpha_n \alpha_j \mathbf{x}^{n^T} \mathbf{x}^j - b \sum_{n=1}^N \alpha_n y^n \quad (15.49)$$

Dado el resultado de la ecuación (15.47), el último término de la expresión (15.49) es igual cero y por tanto:

$$L(\mathbf{w}, b; \boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{j=1}^N y^n y^j \alpha_n \alpha_j \mathbf{x}^{n^T} \mathbf{x}^j \quad (15.50)$$

Como se puede observar la ecuación (15.50) depende únicamente de las variables duales. Finalmente, el problema dual se plantea como el siguiente problema de maximización de la ecuación (15.50):

$$\begin{aligned} \max_{\alpha} \quad & L_D(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{j=1}^N y^n y^j \alpha_n \alpha_j \mathbf{x}^{n^T} \mathbf{x}^j \\ \text{sujeto a} \quad & \sum_{n=1}^N \alpha_n y^n = 0 \\ & \alpha_n \geq 0, \quad n = 1, \dots, N \end{aligned} \quad (15.51)$$

donde el primer conjunto de restricciones viene dado por el resultado obtenido en la ecuación (15.47), mientras que el segundo indica que los multiplicadores de Lagrange deben ser no negativos.

Una vez obtenidos los parámetros óptimos, la forma de designar la clase de cada dato es muy sencilla, ya que bastará con determinar en qué lado de la región de decisión del hiperplano $\mathbf{w}^T \mathbf{x} + b = 0$ se encuentra. Para ello, a cada dato se le asignará la clase resultante de la operación $sgn(\mathbf{w}^T \mathbf{x} + b)$, donde $sgn(\cdot)$ es la función signo.

15.4.2 Caso no linealmente separable

El caso analizado anteriormente, aunque es muy interesante desde un punto de vista teórico, tiene poco interés práctico puesto que es infrecuente encontrar problemas reales en los que los datos sean linealmente separables. Por tanto, es preciso generalizar el problema planteado, permitiendo que algunos datos puedan estar mal clasificados por el hiperplano obtenido. Aceptar algunos errores de clasificación equivale a permitir violaciones en las restricciones del problema planteado. Para ello, se introducen unas nuevas variables ξ_n (*variables de holgura*) en el problema definido en la ecuación (15.44), tanto en la función objetivo como en las restricciones, y se obtiene un nuevo problema de optimización definido como:

$$\begin{aligned} \text{Minimizar} \quad & E(\mathbf{w}, \xi_n) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n \\ \text{sujeto a} \quad & y^n (\mathbf{w}^T \mathbf{x}^n + b) - 1 + \xi_n \geq 0, \quad n = 1, \dots, N \\ & \xi_n \geq 0, \quad n = 1, \dots, N \end{aligned} \quad (15.52)$$

Este nuevo problema de optimización es similar al del caso separable, aunque el nuevo término incluido hace que el clasificador tenga un *margen flexible* (*soft margin*). Las nuevas variables de holgura permiten que algunos ejemplos estén mal clasificados pero, por otro lado, el efecto del término $C \sum_{n=1}^N \xi_n$ en la función a minimizar, es el de penalizar los errores de clasificación para evitar un deterioro importante del rendimiento del sistema. El significado de las variables de holgura puede observarse gráficamente en la Figura 15.16. Las variables ξ_n tomarán valor cero en el caso de que la restricción original se mantenga (es decir, cuando el ejemplo esté en la zona definida por el margen de su clase), y un valor mayor que cero cuando no lo esté.

Esta cantidad vendrá dada en función de la distancia entre el dato mal clasificado y el hiperplano que define el margen de la clase a la que pertenece.

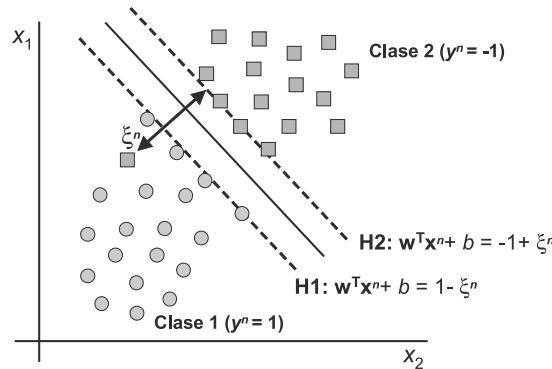


Figura 15.16: Las variables de holgura permiten que algunos ejemplos no queden correctamente clasificados.

Por otro lado, la constante C en el problema planteado, que multiplica al término relativo al coste, actúa como término de compromiso entre el tamaño del margen (término de regularización o penalización de la complejidad) y el error de clasificación:

- Para valores pequeños de C , el primer término de la función a minimizar prevalece, por tanto se logrará un mayor margen a costa de obtener un mayor error de clasificación en el conjunto de datos de entrenamiento.
- Para valores grandes de C , el segundo término prevalece, por lo que se obtiene un menor margen pero también un menor error de clasificación en el conjunto de entrenamiento.

La determinación del valor óptimo del parámetro C para cada conjunto de datos no es una tarea trivial. Una de las técnicas empleadas habitualmente para establecer su valor es la validación cruzada (*cross-validation*) [Efron y Tibshirani, 1993].

Una vez determinado el problema a resolver, éste se puede abordar empleando un planteamiento similar al caso linealmente separable. Por tanto, se definirá la función de Lagrange para este nuevo problema:

$$L(\mathbf{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n - \sum_{n=1}^N \alpha_n [y^n (\mathbf{w}^T \mathbf{x}^n + b) - 1 + \xi_n] - \sum_{n=1}^N \beta_n \xi_n \quad (15.53)$$

donde $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)^T$ y $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_N)^T$ son los multiplicadores no negativos de Lagrange.

Para obtener la solución, se deriva la función de Lagrange respecto a las variables primarias, \mathbf{w} , b y $\boldsymbol{\xi}$, y se igualan las derivadas a cero:

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w} - \sum_{n=1}^N \alpha_n y^n \mathbf{x}^n = 0 \quad (15.54)$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{n=1}^N \alpha_n y^n = 0 \quad (15.55)$$

$$\frac{\partial}{\partial \xi_n} L(\mathbf{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = C - \alpha_n - \beta_n = 0 \quad (15.56)$$

Despejando en las ecuaciones (15.54) y (15.56) se obtiene:

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y^n \mathbf{x}^n, \quad (15.57)$$

$$\beta_n = C - \alpha_n. \quad (15.58)$$

Por tanto, sustituyendo las ecuaciones (15.57) y (15.58) en la función de Lagrange, y maximizando dicha función, se obtiene el siguiente *problema dual*:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & L_D(\boldsymbol{\alpha}) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{j=1}^N y^n y^j \alpha_n \alpha_j \mathbf{x}^{nT} \mathbf{x}^j \\ \text{sujeto a} \quad & \sum_{n=1}^N \alpha_n y^n = 0 \\ & 0 \leq \alpha_n \leq C, \quad n = 1, \dots, N \end{aligned} \quad (15.59)$$

El cual es un problema de programación cuadrática que puede resolverse aplicando los métodos de optimización apropiados.

15.4.3 Máquinas de vectores soporte no lineales

La mayoría de los problemas reales no son linealmente separables, lo que provoca que los clasificadores lineales estén muy limitados. La idea de los clasificadores no lineales consiste en usar un clasificador lineal, como los descritos en los apartados anteriores, pero en un espacio de características, en lugar del espacio original de los datos. Para ello se realiza una transformación de los datos de entrada \mathbf{x} en otro espacio:

$$\mathbf{x} \rightarrow \Phi(\mathbf{x}) = (a_1 \phi_1(\mathbf{x}), a_2 \phi_2(\mathbf{x}), \dots, a_M \phi_M(\mathbf{x})) \quad (15.60)$$

donde las ϕ_i , $i = 1, \dots, M$ son unas funciones no lineales predeterminadas. Como consecuencia de esto, un clasificador lineal en $\Phi(\mathbf{x})$ (espacio transformado) será un clasificador no lineal en el espacio original de \mathbf{x} .

Por tanto, dado el problema dual presentado anteriormente en la ecuación (15.59), para el caso lineal con datos no separables bastará una ligera modificación para adaptarlo al caso no lineal:

$$\begin{aligned} \max_{\alpha} \quad & L_D(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{j=1}^N y^n y^j \alpha_n \alpha_j \Phi(\mathbf{x}^n)^T \Phi(\mathbf{x}^j) \\ \text{sujeto a} \quad & \sum_{n=1}^N \alpha_n y^n = 0 \\ & 0 \leq \alpha_n \leq C, \quad n = 1, \dots, N \end{aligned} \quad (15.61)$$

Como se puede observar, el nuevo problema planteado en la ecuación (15.61) es prácticamente igual al descrito en la ecuación (15.59), excepto que en este caso se ha sustituido el producto de los datos en el espacio original ($\mathbf{x}^n^T \mathbf{x}^j$) por el producto de los correspondientes vectores en el espacio transformado ($\Phi(\mathbf{x}^n)^T \Phi(\mathbf{x}^j)$). La ventaja de este planteamiento es que no es necesario calcular $\Phi(\mathbf{x})$, sino que es suficiente con conocer el producto $\Phi(\mathbf{x}^n)^T \Phi(\mathbf{x}^j)$. Este producto es fácilmente calculable mediante el uso del *truco del núcleo* (del inglés, *kernel trick*). Se trata de definir una *función núcleo* K de la forma:

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^T \Phi(\mathbf{y}) \quad (15.62)$$

que cumpla con las características de 1) producir regiones de decisión interesantes, y 2) que sea sencillo, incluso para grandes dimensiones, calcular el producto vectorial $K(\mathbf{x}, \mathbf{y})$. Una vez definida esta función, trabajar con las transformaciones no lineales en el espacio de características resultará muy sencillo. Por tanto, el entrenamiento de una SVM en el caso no lineal requerirá la elección de:

- La *función núcleo (kernel)*, que determinará la forma de la región de decisión. Algunos ejemplos de las funciones kernel empleadas habitualmente son:

- $K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{\sigma^2}\right)$ de tipo gaussiana.
- $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + \theta)^r$ de tipo polinomial de grado r .
- $K(\mathbf{x}, \mathbf{y}) = \tanh(\mathbf{x}^T \mathbf{y} + \theta)$ de tipo sigmoidea.

- El *parámetro de suavizado (smoothing parameter)* de la función núcleo: la varianza de la función gaussiana, el grado de las funciones polinomiales, etcétera.
- La *constante C*, implicada en la ecuación 15.61, que pondera el coste de los errores de clasificación en la función de error.

15.4.4 Máquinas de Vectores Soporte Multiclas

Hasta el momento se han visto problemas de clasificación binaria (dos clases). Sin embargo, en aplicaciones reales suele ser necesaria la clasificación de los datos en más de dos clases. Por ello, para extender las SVM a problemas multiclas se suelen emplear métodos de votación basados en la combinación de SVM binarias:

- *Uno contra al resto:* Considera el problema multiclase como una colección de problemas dos clases, empleando para ello un clasificador binario para cada una de las clases existentes. Por tanto, si existen S clases se emplearán S clasificadores. Para el i -ésimo clasificador los datos pertenecientes a la clase i se les asigna la etiqueta $+1$ y al resto de datos -1 . Una vez entrenados los S clasificadores, para seleccionar la clase a la que pertenece cada dato se suele emplear un sistema de votación entre todos los clasificadores. El más habitual es el método *ganador de todos (winner-takes-all)*. En este esquema se asigna la etiqueta del clasificador que tiene una mayor certeza de que el dato (\mathbf{x}) está en la clase $+1$, es decir:

$$f(\mathbf{x}, \mathbf{w}) = \arg \max_i (f_i(\mathbf{x}, \mathbf{w})),$$

donde $f_i(\mathbf{x}, \mathbf{w})$ es la salida del clasificador i .

- *Uno contra uno:* En este método se construyen $S(S - 1)/2$ clasificadores separando cada uno de ellos una de las clases de otra diferente. Para cada par (i, j) de etiquetas de clase, tal que $i < j$, se construye un clasificador binario, clasificando los puntos de la clase i con la etiqueta $+1$ y los de la clase j con la etiqueta -1 . Una vez entrenados los clasificadores y, similarmente al caso anterior, se emplea un método de votación entre ellos para seleccionar la clase final de cada dato.

15.5 Resumen

En este capítulo se han analizado las características básicas de las redes de neuronas artificiales. El objetivo de éstas es el de encontrar una función capaz de explicar y reproducir unos datos observados, principalmente, en problemas de regresión y clasificación. Para ello, los modelos neuronales deben ajustar los valores de sus parámetros de acuerdo a conjunto de datos representativos del problema y un algoritmo de aprendizaje. Se han explicado los dos tipos básicos de aprendizaje existentes (supervisado y no supervisado) y, dentro de ellos, las arquitecturas de redes más comunes. Así, se presentaron el perceptrón, el perceptrón multicapa y las redes de base radial, como arquitecturas de aprendizaje supervisado, y los mapas autoorganizativos como principal arquitectura no supervisada. Por último, se presentaron las máquinas de vectores soporte que, aunque no están consideradas redes de neuronas propiamente dichas, también proporcionan un modo de encontrar modelos subyacentes a un conjunto de datos mediante un proceso de ajuste de parámetros. No se han comentado en este capítulo detalles importantes que influirán en el rendimiento de una red y que determinan ciertas cuestiones sobre cómo preparar los datos de entrenamiento, condiciones que ha de cumplir el proceso de aprendizaje para ser fiable o cómo determinar el error de una red entrenada. Para profundizar en estas y otras cuestiones remitimos al lector a la bibliografía recomendada.

15.6 Ejercicios propuestos

15.1. Considerar dos clases unidimensionales con distribución gaussiana, C_1 y C_2 , que tienen la misma varianza igual a 1. Sus medias son:

$$\begin{aligned}\mu_1 &= -10 \\ \mu_2 &= +10.\end{aligned}$$

Estas clases son linealmente separables. Diseñar un clasificador basado en un perceptrón que separe ambas clases.

15.2. Emplear el perceptrón simple para realizar las funciones lógicas: AND, OR, COMPLEMENTO y NAND₂. La función NAND₂, también conocida como AND negada de dos entradas, devuelve un 0 si ambas entradas son 1, y devuelve 1 en caso contrario. Una de las limitaciones básicas del perceptrón es que no puede implementar el OR exclusivo. Explicar la razón de dicha limitación.

15.3. Implementar un perceptrón multicapa con el algoritmo de *backpropagation* en el lenguaje de programación que desees. En algunos entornos, como Matlab, hay funciones implementadas que te pueden facilitar esta labor. Con el perceptrón desarrollado:

1. Determinar si una secuencia de 6 bits de longitud es simétrica alrededor de su centro. Para ello, generar un conjunto de entrenamiento de pares entrada/salida.
2. Aproximar la función $\sin(x+?y)$. En este caso, generar un conjunto de datos de entrenamiento dentro del intervalo $[0, 2\pi]$.

Para cada uno de los puntos anteriores, determinar, justificando cada elección:

- La arquitectura de red,
- la(s) función(es) de activación utilizada(s),
- los valores para cada uno de los parámetros de la red.

Comprobar la capacidad de generalización de cada perceptrón. Incorporar para el análisis del punto (2) el gráfico de la función y el de la aproximación obtenida.

15.4. Tal como se explica en la sección 15.2.3, existen distintas posibilidades para las funciones base de una Red de Base Radial. En general, se demuestra que, bajo ciertas condiciones, el resultado de la red es insensible a la forma concreta de estas funciones. Comprobar experimentalmente este resultado generando para ello un conjunto de entrenamiento formado por 3 clases de datos bidimensionales provenientes de 3 mezclas distintas de distribuciones gaussianas. Con estos datos entrenar distintas redes con las siguientes funciones base (todas ellas tienen la propiedad $\phi(x) \rightarrow 0$ cuando $x \rightarrow 0$):

a) Gaussiana: $\phi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right)$

- b) Multicuadrática: $\phi_j(\mathbf{x}) = (\|\mathbf{x} - \boldsymbol{\mu}_j\|^2 + \sigma_j^2)^{1/2}$
- c) *Thin plate spline*: $\phi_j(\mathbf{x}) = \|\mathbf{x} - \boldsymbol{\mu}_j\|^2 \ln(\|\mathbf{x} - \boldsymbol{\mu}_j\|^2)$
- d) Lineal: $\phi_j(\mathbf{x}) = \|\mathbf{x} - \boldsymbol{\mu}_j\|$
- e) Cúbica: $\phi_j(\mathbf{x}) = \|\mathbf{x} - \boldsymbol{\mu}_j\|^3$

Representar gráficamente los datos de las 3 clases del conjunto de entrenamiento, así como las trayectorias de evolución de los centros $\boldsymbol{\mu}_j$ de las funciones base durante el entrenamiento.

15.5. Generar un problema de regresión creando un conjunto de entrenamiento a partir de una mezcla de distintas distribuciones gaussianas. Utilizando también como función base una función gaussiana, entrenar distintas RBFN para investigar la relación entre los siguientes aspectos y cómo influyen en los resultados de la red:

- El número de funciones base o neuronas ocultas,
- la varianza σ_j de dichas funciones y
- el número de funciones gaussianas utilizadas para la generación de los datos de entrenamiento y su varianza.

Utilizar diagramas del estilo de la Figura 15.6 para ilustrar los resultados.

15.6. Dada una Máquina de Vectores Soporte de tipo lineal empleada para clasificar los datos de dos clases (la clase 1 representada por triángulos y la clase 2 representada por círculos); indicar, en los casos que se muestran en la Figura 15.17, cuáles serían los vectores soporte y dibujar las rectas H_1 y H_2 que definen el margen del clasificador.

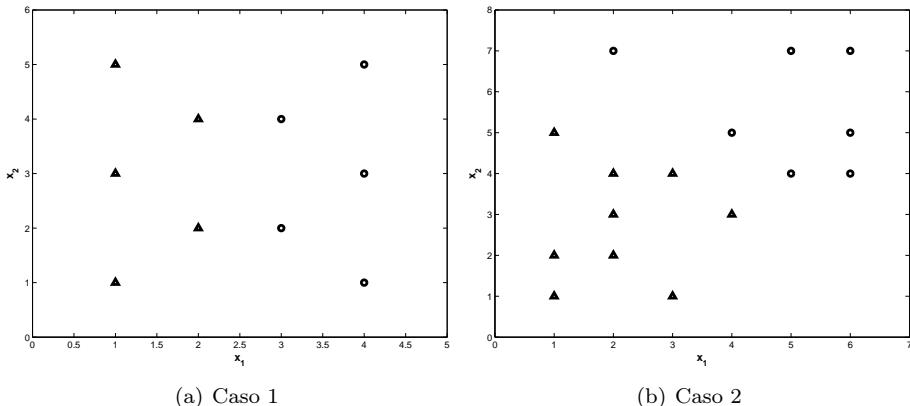


Figura 15.17: Datos del Ejercicio 6.

15.7. Indicar cuáles de las siguientes afirmaciones son correctas en el caso de una Máquina de Vectores Soporte de tipo lineal:

- El número de vectores soporte se determina antes del proceso de entrenamiento.
- La solución obtenida no depende del orden de los datos en el conjunto de entrenamiento.
- Los vectores soporte son algunos de los datos del conjunto de entrenamiento.
- Independientemente de si el problema es linealmente separable o no, no existe ningún dato dentro del margen definido por los hiperplanos H_1 y H_2 .
- Existe, al menos, un vector soporte para cada una de las clases del problema a resolver.
- La solución del problema sólo puede obtenerse empleando la formulación dual.

Referencias

- ALMEIDA, L. B.; LANGLOIS, T.; AMARAL, J. D. y PLAKHOV, A.: «Parameter adaptation in stochastic optimization». En: D. Saad (Ed.), *On-line Learning in Neural Networks*, capítulo 6, pp. 111–134. Cambridge University Press, 1999.
- BISHOP, C. M.: *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, USA, 1995.
- CASTILLO, E.; FONTENLA-ROMERO, O.; BETANZOS, A. ALONSO y GUIJARRO-BERDIÑAS, B.: «A Global Optimum Approach for One-Layer Neural Networks». *Neural Computation*, 2002, **14**(6), pp. 1429–1449.
- CASTILLO, E.; GUIJARRO-BERDIÑAS, B.; FONTENLA-ROMERO, O. y ALONSO-BETANZOS, A.: «A Very Fast Learning Method for Neural Networks Based on Sensitivity Analysis». *Journal of Machine Learning Research*, 2006, **7**(Jul), pp. 1159–1182.
- CHEN, S.; COWAN, C.F. y GRANT, P.M.: «Orthogonal least squares learning algorithm for radial basis function networks». *IEEE Transactions on Neural Networks*, 1991, **2**(2), pp. 302–309.
- CHERKASSKY, V. y MULIER, F.: *Learning from Data: Concepts, Theory, and Methods*. Wiley, New York, 1998.
- CRISTIANINI, N. y SHawe-Taylor, J.: *An introduction to Support Vector Machines and other Kernel-based learning methods*. Cambridge University Press, Cambridge, 2000.
- CYBENKO, G.: «Aproximation by superpositions of a sigmoidal function». *Mathematics of Control, Signals and Systems*, 1989, **2**, pp. 304–314.
- DEMPSTER, A. P.; LAIRD, N. M. y RUBIN, D. B.: «Maximum Likelihood from Incomplete Data via the EM algorithm». *Journal of the Royal Statistical Society*, 1977, **B 39**(1).
- DENNIS, J. E. y SCHNABEL, R. B.: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1983.
- EFRON, B. y TIBSHIRANI, R.: *An introduction to bootstrap*. Chapman & Hall, 1993.
- FONTENLA-ROMERO, O.; ERDOGMUS, D.; PRINCIPE, J.C.; ALONSO-BETANZOS, A. y CASTILLO, E.: «Linear least-squares based methods for neural networks learning». *Lecture Notes in Computer Science*, 2003, **2714(84–91)**.
- GIROSI, F. y POGGIO, T.: «Networks and the best approximation property». *Biological Cybernetics*, 1990, **63**, pp. 169–176.
- HAGAN, M. T. y MENHAJ, M.: «Training feedforward networks with the Marquardt algorithm». *IEEE Transactions on Neural Networks*, 1994, **5**(6), pp. 989–993.

- HARTMAN, E.; KEELER, J. D. y KOWALSKI, J. M.: «Layered neural networks with gaussian hidden units as universal approximators». *Neural Computation*, 1990, **2**(2), pp. 210–215.
- HAYKIN, S.: *Neural Networks. A Comprehensive Foundation*. Prentice-Hall, New Jersey, USA, 2^a edición, 1999.
- HEBB, D. O.: *The organization of behaviour*. John Wiley and Sons, New York, 1949.
- HUSH, D. R. y SALAS, J. M.: «Improving the learning rate of back-propagation with the gradient reuse algorithm». *Proceedings of the IEEE Conference of Neural Networks*, 1988, **1**, pp. 441–447.
- JACOBS, R. A.: «Increased Rates of Convergence Through Learning Rate Adaptation». *Neural Networks*, 1988, **1**(4), pp. 295–308.
- KOHONEN, T.: *Self-Organizing Feature Maps*. Springer-Verlag, New York, 1995.
- LECUN, Y.; BOTTOU, L.; ORR, G.B. y MÜLLER, K.-R.: «Efficient BackProp». En: G.B. Orr y K.R. Müller (Eds.), *Neural Networks: Tricks of the trade*, Número 1524 en LNCS. Springer-Verlag, 1998.
- LINSKER, R.: «Self-organization in a perceptual network». *Computer*, 1988, **21**, pp. 105–117.
- MASTERS, T.: *Signal and Image Processing with Neural Networks*. John Wiley & Sons, 1994.
- MCCULLOCH, W. S. y PITTS, W. H.: «A logical calculus of the ideas immanent in nervous activity». *Bulletin of Mathematical Biophysics*, 1943, **5**, pp. 115–133.
- MINSKY, M. L. y PAPERT, S. A.: *Perceptrons*. MIT Press, Cambridge, 1969.
- MOLLER, M. F.: «A scaled conjugate gradient algorithm for fast supervised learning». *Neural Networks*, 1993, **6**, pp. 525–533.
- MOODY, J. y DARKEN, C.J.: «Fast learning in networks of locally-tuned processing units». *Neural Computation*, 1989, **1**(2), pp. 281–294.
- NGUYEN, D. y WIDROW, B.: «Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights». *Proceedings of the International Joint Conference on Neural Networks*, 1990, **3**, pp. 21–26.
- ORR, G. B. y LEEN, T. K.: «Using Curvature Information for Fast Stochastic Search». En: M.I. Jordan; M.C. Mozer y T. Petsche (Eds.), *Neural Information Processing Systems*, volumen 9, pp. 606–612. MIT Press, Cambridge, 1996.
- PARK, J. y SANDBERG, I. W.: «Universal approximation using radialbasis function networks». *Neural Computation*, 1991, **3**(2), pp. 246–257.

- POWELL, M. J. D.: «Restart procedures for the conjugate gradient method». *Mathematical Programming*, 1977, **12**, pp. 241–254.
- ROSENBLATT, F.: *Principles of Neurodynamics: Perceptron and theory of brain mechanisms*. Spartan, Washington DC, 1962.
- SCHRAUDOLPH, N.: «Fast Curvature Matrix-Vector Products for Second Order Gradient Descent». *Neural Computation*, 2002, **14(7)**, pp. 1723–1738.
- SMOLA, A. J. y SCHÖLKOPF, B.: *Learning with kernels*. MIT Press, 2002.
- SWINGLER, K.: *Applying Neural Networks: A Practical Guide*. Morgan Kaufmann, 1996.
- VAPNIK, V.: *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.
- VAPNIK, V.: *Statistical Learning Theory*. John Wiley and Sons, Inc., New York, 1998.
- VOGL, T. P.; MANGIS, J. K.; RIGLER, A. K.; ZINK, W. T. y ALKON, D. L.: «Accelerating the Convergence of Back-Propagation Method». *Biological Cybernetics*, 1988, **59**, pp. 257–263.
- WEIR, M. K.: «A method for self-determination of adaptive learning rates in back propagation». *Neural Networks*, 1991, **4**, pp. 371–379.
- WIDROW, B. y HOFF, M. E.: «Adapting switching circuits». In *IRE WESCON Convention Record*, 1960, **4**, pp. 96–104.
- YAM, J. Y. F.; CHOW, T. W. S y LEUNG, C. T: «A New method in determining the initial weights of feedforward neural networks». *Neurocomputing*, 1997, **16(1)**, pp. 23–32.

Capítulo 16

Técnicas de agrupamiento

Amparo Vila Miranda y Miguel Delgado Calvo-Flores

Universidad de Granada

16.1 Introducción

En una primera aproximación, y acudiendo a la definición clásica, las técnicas de agrupamiento (*clustering*) se pueden definir como:

Una técnica de clasificación no supervisada de *elementos* (observaciones, datos o vectores de características) en grupos (clusters).

Dado que la clasificación es una de las abstracciones básicas en la inducción de conocimiento, este problema ha sido tratado en muchos contextos y por investigadores de muchas disciplinas (Biología, Psicología, Análisis Económico, Sociología, etc.), de forma que el término agrupamiento se usa en numerosas comunidades de investigación para describir el proceso de clasificar en grupos, un conjunto de elementos sin tener una información previa acerca de su estructura. Estas comunidades tienen diferente terminología para describir los componentes del proceso y diversas metodologías para resolver los problemas que el agrupamiento presenta.

El agrupamiento ha sido ampliamente estudiado desde hace más de cuarenta años. Desde finales de los años 60, se han desarrollado técnicas de agrupamiento dentro el ámbito del Análisis de Datos y de la Taxonomía Numérica. Posteriormente, las técnicas de agrupamiento se ha incluido dentro del campo de la IA, encuadrándose en el área del Aprendizaje no Supervisado. Por último, la Minería de Datos también incluye dentro de sus técnicas las técnicas de agrupamiento, recuperando las técnicas y metodologías previamente desarrolladas, extendiéndolas y adaptándolas al volumen de datos que se procesan en este campo.

Nos encontramos pues con un tema muy desarrollado, tanto por el largo tiempo que lleva estudiándose como por la amplia variedad de los enfoques teóricos empleados y de los campos donde se ha aplicado. Por ello no es posible dar una visión muy detallada del tema en un libro de tipo generalista como el que nos ocupa. Así pues, en este

capítulo nos centraremos más en presentar las ideas básicas acerca del agrupamiento que en estudiar en profundidad los matices y variaciones de diversos algoritmos. Para ello remitimos al lector interesado a textos específicos, algunos de los cuales se recogen en la bibliografía [Berkhin, 2002; Jain y otros, 1999; Jain y Dubes, 1988].

El capítulo está organizado de la siguiente manera: comenzaremos con los conceptos básicos, definiendo y encuadrando problemas y técnicas. Posteriormente, estudiaremos los problemas asociados a la representación de datos y las medidas de similaridad más habituales, que son el punto de partida para el agrupamiento. Seguidamente, analizaremos las técnicas más clásicas de agrupamiento jerárquico y particional, comentando después técnicas más actuales, haciendo especial mención a las que se han adaptado para resolver problemas de Minería de Datos. Terminaremos con los métodos que contemplan grupos no exclusivos basados en Lógica Borrosa (véase el capítulo 7).

16.2 Conceptos básicos

Como se ha dicho, el proceso de agrupamiento (*cluster analysis*) no es más que la organización de una colección de elementos en un conjunto de grupos homogéneos. Habitualmente estos elementos están representados por un vector de valores de atributos, es decir, son puntos de algún espacio multidimensional. Estos valores también se suelen denominar *atributos*, *componentes* o simplemente variables. Intuitivamente, dos elementos pertenecientes a un agrupamiento válido deben ser más parecidos entre sí que aquellos que estén en grupos distintos. Partiendo de esta idea se desarrollan las técnicas de agrupamiento. Obviamente, estas técnicas dependen de cómo sean los datos de partida, de qué medidas de semejanza se estén utilizando y de qué clase de problemas se estén resolviendo. Por ejemplo, no es lo mismo clasificar bacterias según una taxonomía jerárquica que agrupar los pixels de una imagen por colores.

También es importante distinguir entre clasificación supervisada (dentro de la cual se encuentra el análisis discriminante) y clasificación no supervisada (dentro de la cual se incluyen las técnicas de agrupamiento) y clasificación supervisada. En el primer caso, se posee la información de a qué clase pertenece cada elemento, y lo que se desea es determinar cuáles son los atributos que intervienen en la definición de las clases y qué valores de los mismos determinan estas. En los capítulos 15 y 17 se pueden analizar algunas técnicas de clasificación supervisada. En caso de la clasificación no supervisada, no tenemos ninguna información acerca de organización de los elementos en grupos o clases y, por lo tanto, el objetivo es encontrar dicha organización en base a la proximidad entre los elementos. No existe apenas información previa acerca de esta estructura, y la interpretación de las clases o grupos obtenidos es una labor a realizar “a posteriori” por parte del analista. Un ejemplo clásico de agrupamiento sería la búsqueda de grupos de clientes de una entidad bancaria utilizando para ello datos de la cuenta corriente: edad, dirección, nivel de renta, etc. Un ejemplo de clasificación sería encontrar los elementos que determinan la aparición de cáncer de pulmón analizando datos de edad, calidad de vida, nivel económico, etc., tanto de personas enfermas como sanas.

Los problemas y metodologías para el agrupamiento pueden clasificarse según distintos criterios. En la Figura 16.1, tomada de [Jain y otros, 1999] y [Lance y Williams, 1967], se puede ver un árbol con las distintas técnicas de agrupamiento. Comentaremos brevemente esta clasificación, y otras que se relacionan y superponen a ellas, ya que constituyen el hilo conductor de este capítulo:

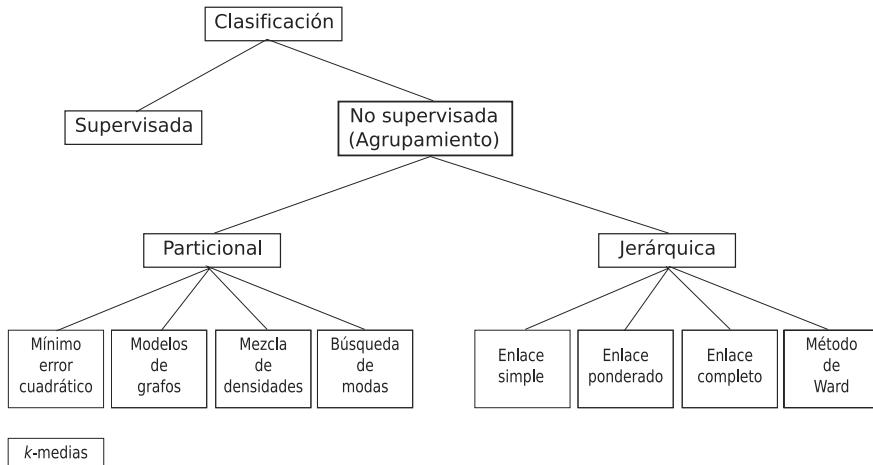


Figura 16.1: Árbol de tipos de métodos de clasificación.

- **Clasificación supervisada y no supervisada.** Como hemos comentado la clasificación es el proceso mediante el cual se agrupa un conjunto de elementos en función de una representación de los mismos en un espacio n -dimensional. En el caso de que sea supervisada se tiene información acerca de los grupos de manera que sabemos a qué grupo pertenece cada clase, entonces lo que se desea es encontrar un conjunto de “criterios”, probablemente reglas, que nos permitan, dado un nuevo elemento, situarlo en un grupo (para más información véase los capítulos 15, 17 y 18). En el caso de la clasificación no supervisada no se tiene mucha información acerca de los grupos, a veces no se sabe siquiera cuántos grupos hay, se trata entonces de encontrar un agrupamiento que reúna en un mismo grupo los elementos más “parecidos” y coloque en grupos diferentes a los elementos “dispareos”.

Como vemos hay una gran diferencia de enfoque entre ambos problemas; de hecho, en muchos casos el agrupamiento puede verse como el primer paso para la clasificación supervisada.

- **Agrupamiento particional y agrupamiento jerárquico.** El objetivo final del proceso de agrupamiento es obtener un conjunto de clases o grupos. Cuando estos grupos son disjuntos y cubren todo el conjunto de elementos se dice que el agrupamiento es *particional*. En algunos casos lo que se desea no es exactamente un agrupamiento particional sino jerarquía de agrupamientos particionales

“anidados”, de tal manera que cada grupo de un nivel se divide en varios en el nivel siguiente. Esta estructura se denomina *agrupamiento jerárquico* y tiene una representación gráfica muy intuitiva denominada *dendrograma*.

Las técnicas de agrupamiento jerárquico son muy populares en ciencias biológicas, sociales y de la conducta, donde se hace necesario construir taxonomías. Las técnicas particionales se usan fundamentalmente en aplicaciones de ingeniería donde se necesitan particiones simples. El agrupamiento particional es particularmente útil en el caso de que se trabaje con grandes bases de datos, como en el caso de las aplicaciones de Minería de Datos, ya que los dendrogramas son poco prácticos cuando manejan más de unos pocos cientos de elementos.

Dependiendo de cómo se planteen los objetivos del agrupamiento, pueden aparecer distintos enfoques, tanto en el caso particional como en el jerárquico. Estos enfoques se representan en las distintas hojas del árbol anterior y realmente se identifican con la forma en que se contemplen o interpreten tanto los elementos como los grupos.

- Si se considera que los datos están representados mediante un grafo donde los vértices son los elementos y las aristas son conexiones entre ellos, definidas a través de alguna función de semejanza, entonces los grupos aparecen como componentes conectadas del grafo. Este enfoque es el que hemos denominado *Modelos de Grafos*, y aparece tanto en caso de agrupamiento particional como en el jerárquico, ya que los agrupamientos jerárquicos de enlace, simple, completo y medio se encuadran también dentro de este enfoque. Una filosofía similar sigue el bien conocido método del *vecino más cercano*.
- Cuando se considera que los grupos deben ser “cohesionados” de manera que los elementos de un mismo grupo estén más cercanos entre sí y la distancia entre grupos sea la mayor posible, aparece una amplia familia de modelos de agrupamiento particional. Uno de los más extendidos es el de mínimos cuadrados, donde el criterio de cohesión se obtiene como la suma total de la distancia de cada elemento al punto medio (*centroide*) del grupo al que pertenece, este valor obviamente debe ser mínimo. El bien conocido *método de las k-medias* se encuadra en esta categoría.
- Cuando se considera que un grupo es una región del espacio n -dimensional donde la densidad de elementos es muy alta, rodeada de una zona de baja densidad, aparecen los denominados métodos basados en análisis de densidad. Dentro de este enfoque se encuadran los métodos de *estimación de densidad* y de *búsqueda de modas*, cuya idea de base es emplear estimaciones estadísticas de la densidad de probabilidad de cada grupo, suponiendo que cada uno de ellos es una muestra representativa de una población. La misma filosofía sigue el método que hemos denominado de *mezcla de densidades*.

La clasificación de métodos de agrupamiento que presenta la Figura 16.1 es bastante exhaustiva pero no es la única división posible, y desde el punto de vista de la relación entre los diferentes agrupamientos o de la forma de obtenerlos podemos hablar de:

- **Agrupamientos exclusivos, agrupamientos no exclusivos y agrupamientos borrosos.** Todos los enfoques comentados en los párrafos anteriores parten de la hipótesis de no-solapamiento, es decir que deseamos un agrupamiento donde cada punto pertenezca sólo a un grupo. Cuando se relaja esta hipótesis aparecen métodos de agrupamiento que admiten solapamiento o no-exclusivos. Los métodos de agrupamiento no-exclusivos que han tenido más éxito son los que suponen que los grupos son *conjuntos borrosos* de forma que un elemento puede pertenecer a diversos grupos con un grado de pertenencia a cada uno.
- **Agrupamientos aglomerativos y divisivos.** Este aspecto se refiere fundamentalmente a la estructura del algoritmo que desarrolla el método. Si se parte de un agrupamiento en el que cada elemento es un grupo y se van construyendo nuevas soluciones uniendo grupos en otros más amplios, se tiene un algoritmo de tipo *aglomerativo*, si el proceso es el contrario, dividiendo el espacio total en grupos que sucesivamente se van haciendo menores y más compactos tendremos un algoritmo *divisivo*.

Otras posibles clasificaciones se pueden encontrar en [Jain y Dubes, 1988]. Obviamente una técnica concreta reúne un conjunto de características de acuerdo con las divisiones antes citadas. Por ejemplo, la mayoría de las técnicas de agrupamiento jerárquico se han recogido bajo el acrónimo de técnicas SHAN (Secuencial, Aglomerative, Hierarchical, Nonoverlapping; Secuencial, Aglomerativo, Jerárquico y Sin solapamiento).

16.3 Los datos de partida

Los algoritmos de agrupamiento reúnen los elementos basándose en índices de proximidad (o distancia) entre ellos, pero la información de partida puede estar representada de dos formas:

- Por medio de una *matriz de elementos*, donde los n elementos de un determinado conjunto se representan por una serie de m medidas (atributos, puntuaciones, etc.) mediante un vector de m dimensiones, quedando el conjunto representado como una matriz $\mathcal{X} = [x_{ij}]$ de dimensiones $n \times m$, donde cada fila \mathbf{x}_i de esta matriz representa un elemento y cada columna \mathbf{y}_j se denomina *atributo* o medida.
- Por medio de una *matriz de proximidad*, \mathcal{D} , de dimensión $n \times n$, siendo n el número de elementos, donde el valor de la casilla \mathcal{D}_{ik} representa una medida de la proximidad (distancia) entre el elemento \mathbf{x}_i y el \mathbf{x}_k . Habitualmente la matriz de proximidad se calcula a partir de la matriz de elementos, pero en ciertas aplicaciones psicométricas y sociológicas es posible que los datos se recojan directamente en términos de concordancias y se tenga de partida una matriz de proximidad.

En los siguientes párrafos comentamos los problemas de obtención de ambas matrices.

16.3.1 La matriz de elementos

El problema de elegir una representación adecuada de los datos de partida para obtener un agrupamiento no es en absoluto sencillo. De hecho, la preparación de datos es una de las tareas más importantes en todo proceso de extracción de conocimiento y puede dar origen a muchos errores. En [Pang y otros, 2006] se puede encontrar un capítulo completo dedicado a la preparación de datos.

Existen distintos tipos de atributos para representar un elemento. No hay que olvidar que un elemento puede ser un objeto físico (persona, pixel, etc.) o un ente abstracto (estilos de escritura, opiniones políticas, etc.) y que cualquier medida entre estos puede ser usada para agruparlos. Tenemos medidas que pueden ser cuantificadas (atributos cuantitativos) y medidas que no representan forma alguna de cantidad (atributos cualitativos) y ambas divisiones presentan a su vez distintos tipos según la clase de valores que presentan:

- **Los atributos cuantitativos** se dividen en:

- *Atributos con valores continuos*, por ejemplo el peso de una persona, o el nivel de sodio en un suelo.
- *Atributos con valores discretos*, por ejemplo el número de ordenadores de un centro. Un caso importante particular de estos son *atributos binarios* que sólo toman los valores 0 ó 1 y que representa la presencia o ausencia de una determinada característica.
- *Atributos con valores definidos sobre intervalos*, por ejemplo la duración de un suceso.

- **Los atributos cualitativos** se dividen en:

- *Atributos con valores nominales o no ordenados*, por ejemplo el color de un suelo, o el diagnóstico de un enfermo.
- *Atributos con valores ordinales*, por ejemplo el rango de un militar o el nivel de gravedad de un enfermo.

No siempre es posible elegir los datos que intervienen en nuestro problema, y reconocer su tipo es esencial para diseñar una medida de proximidad adecuada e interpretar los resultados del agrupamiento. Es necesario mencionar que existen muchos trabajos acerca de la selección de los atributos para procesos de clasificación supervisada, pero que para agrupamiento se hace necesario un proceso de ensayo-error, ya que no se tiene ninguna información a priori para llevar a cabo un filtrado de dichos atributos.

16.3.2 Índices de proximidad: distancias y semejanzas

En este apartado presentamos un resumen de las formas más conocidas de construir una matriz de proximidad a partir de una matriz de elementos. Para más detalles acerca de este tema se recomienda consultar [Duran y Odell, 1974; Jain y Dubes, 1988]. En primer lugar definiremos el concepto de *Índice de proximidad*.

Definición 16.1. *Índice de proximidad.* Sea I un conjunto de n elementos, decimos que $d : I \times I \rightarrow R$ es un índice de proximidad si y sólo si se verifica:

1. (a) $\forall i \in I d(i, i) = 0$ (en el caso de medidas de distancia)
- (b) $\forall i \in I d(i, i) \geq \max_{k \in I} d(i, k)$ (en el caso de medidas de similaridad)
2. $\forall i, k \in I, d(i, k) = d(k, i)$
3. $\forall i, k \in I, d(i, k) \geq 0$

Los índices de proximidad generalizan conceptos conocidos, como los de *Distancia* o *Índice de semejanza* algunos de los cuales pasamos a explicar.

Definición 16.2. *Distancia.* Se dice que un índice de proximidad d es una distancia si y sólo si se verifica:

1. Las propiedades 1.a, 2 y 3 de la definición 16.1
2. $\forall i, l, k \in I, d(i, k) \leq d(i, l) + d(l, k)$

Las distancias son especialmente utilizadas en el caso de atributos cuantitativos continuos. En la Tabla 16.1 se pueden ver las funciones de distancia más habituales. La distancia Euclídea es la más intuitiva y trabaja muy bien cuando se tienen grupos “compactos” y “aislados”. La distancia de Mahalanobis generaliza la distancia euclídea, mientras que la distancia de Minkowski generaliza a todas las demás. El principal inconveniente de las métricas de Minkowski es que, en general, todas ellas dan un gran peso a los atributos con valores muy grandes, problema que se soluciona con una normalización. Otro problema que presentan los atributos continuos es la posible existencia de correlación entre ellos, lo que se puede paliar utilizando la distancia de Mahalanobis o reduciendo previamente el espacio por medio de un análisis factorial que nos devuelva un nuevo conjunto de atributos independientes. Para más detalles véase [Jain y Dubes, 1988].

NOMBRE	EXPRESIÓN
Euclídea o norma- l_2	$d_2(\mathbf{x}_i, \mathbf{x}_k) = \sqrt{\sum_{j=1}^m (x_{ij} - x_{kj})^2} = [(\mathbf{x}_i - \mathbf{x}_k)^T (\mathbf{x}_i - \mathbf{x}_k)]^{1/2}$
Manhattan o norma- l_1	$d_1(\mathbf{x}_i, \mathbf{x}_k) = \sum_{j=1}^m x_{ij} - x_{kj} $
Norma del supremo	$d_\infty(\mathbf{x}_i, \mathbf{x}_k) = \sup_{j \in \{1, 2, \dots, m\}} x_{ij} - x_{kj} $
Minkowski o norma- l_p	$d_p(\mathbf{x}_i, \mathbf{x}_k) = \sqrt[p]{\sum_{j=1}^m x_{ij} - x_{kj} ^p}$
Distancia de Mahalanobis	$d_M = [(\mathbf{x}_i - \mathbf{x}_k)^T \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_k)]^{1/2}$ Σ es la covarianza muestral o una matriz de covarianza intra-grupos

Tabla 16.1: Distintas funciones de distancia.

NOMBRE	EXPRESIÓN
Índice de Jaccard	$\frac{n_{IK}}{n_{IK} + n_{iK} + n_{Ik}}$
Índice de acoplamiento simple	$\frac{n_{IK}}{n_{IK} + n_{ik}}$
Índice de Russell	$\frac{m}{n_{IK}}$
Índice de Dice	$\frac{2n_{IK}}{2n_{IK} + n_{iK} + n_{Ik}}$
	$\frac{2(n_{IK} + n_{ij})}{m + n_{iK} + n_{Ik}}$
	$\frac{n_{IK}}{n_{IK} + 2(n_{iK} + n_{Ik})}$
	$\frac{(n_{IK} + n_{ik})}{m + n_{iK} + n_{Ik}}$

Tabla 16.2: Distintos índices de semejanza.

Existen también funciones de distancia basadas en la distancia de dos distribuciones de probabilidad y funciones de distancia basadas en el coeficiente de correlación que se aplican al espacio de atributos, no al de elementos.

Definición 16.3. Índice de semejanza Se dice que un índice de proximidad s es una función de semejanza, si y sólo si se verifica:

1. $\forall i \in I \quad d(i, i) = 1$
2. Las propiedades 2 y 3 de la definición 16.1.

Obviamente se puede obtener un índice de semejanza a partir de una distancia, tomando:

$$\forall i, k \in I \quad s(i, k) = 1 - \left(\frac{d(i, k)}{D} \right) \text{ siendo } D = \max_{i, k \in I, i \neq k} (d(i, k))$$

La mayoría de los índices de semejanza, no basados en distancia, se han definido para elementos cuyos atributos son binarios, es decir aquellos que sólo toman valores 0 o 1 y que sirven para reflejar la presencia o ausencia de una determinada característica. La Tabla 16.2 nos muestra los más utilizados con la siguiente notación (para \mathbf{x}_i y \mathbf{x}_k , elementos formados por m variables binarias):

- n_{IK} número de atributos que toman el valor 1 en \mathbf{x}_i y \mathbf{x}_k
- n_{ik} número de atributos que toman el valor 0 en \mathbf{x}_i y \mathbf{x}_k
- n_{iK} número de atributos que toman el valor 0 en \mathbf{x}_i y 1 en \mathbf{x}_k
- n_{ik} número de atributos que toman el valor 1 en \mathbf{x}_i y 0 en \mathbf{x}_k

Un índice de semejanza bastante utilizado cuando se trabaja con documentos es la denominada medida del coseno. Se parte de la representación de cada documento como un vector de frecuencias de aparición de términos. De forma que si $\mathbf{t}_i = (t_{i1}, \dots, t_{id})$ y $\mathbf{t}_j = (t_{j1}, \dots, t_{jd})$ son dos vectores que describen documentos en un espacio d -dimensional a partir de las frecuencias de repetición de una serie de términos, entonces una posible medida de semejanza entre ellos puede ser:

$$\cos(\mathbf{t}_i, \mathbf{t}_j) = \frac{(\mathbf{t}_i \odot \mathbf{t}_j)}{|\mathbf{t}_i||\mathbf{t}_j|}$$

teniendo en cuenta que \odot representa el producto escalar y $|.|$ el módulo, la expresión anterior se reduce a

$$\cos(\mathbf{t}_1, \mathbf{t}_2) = \frac{\sum_{j=1}^d t_{1j}t_{2j}}{\sqrt{\sum_{j=1}^d t_{1j}^2} \sqrt{\sum_{j=1}^d t_{2j}^2}} \quad (16.1)$$

Tanto las distancias como las semejanzas se utilizan para obtener la matriz de proximidad de un conjunto de elementos que es el punto de partida para un proceso de agrupamiento. Ahora bien, cada uno de los enfoques corresponde a un tipo de atributo, claramente las distancias se utilizarán en presencia de atributos continuos, y pueden usarse con valores enteros e incluso ordinales asimilables a enteros. Las semejanzas son adecuadas cuando se trabaje con atributos binarios y pueden utilizarse con atributos nominales no ordinales transformándolos en un conjunto de atributos binarios. Es importante tener en cuenta que puede ser problemático mezclar ambos enfoques directamente, cuando se tienen los dos tipos de atributos. No obstante se han propuesto técnicas mixtas tal como la que se presenta en [Wilson y Martinez, 1997]. A partir de ahora supondremos que tenemos construida la matriz de proximidad. Se puede encontrar un análisis acerca de tipos de datos, cambios de escala, etc, en [Jain y Dubes, 1988; Pang y otros, 2006].

16.4 Técnicas de agrupamiento jerárquico

16.4.1 Ideas básicas

Como hemos comentado anteriormente, un agrupamiento jerárquico es una sucesión de particiones “anidadas” donde cada grupo de elementos perteneciente a una determinada partición está totalmente incluido en algún grupo de la partición siguiente, esta estructura tiene una representación gráfica muy intuitiva que se denomina *Dendrograma*. En la Figura 16.2 se muestra el agrupamiento jerárquico de 10 elementos, cuya matriz de proximidad se refleja en la Tabla 16.3, obtenido por el método de enlace completo que describiremos posteriormente. El agrupamiento se describe por medio del dendrograma donde se presenta cómo se van uniendo los distintos elementos en

grupos: partiendo de una primera partición donde se consideran todos los elementos aislados, en una primera etapa se agrupan los elementos 2 y 3, 7 y 8 , y 9 y 10, en la segunda etapa se agrupan los elementos 4 y 5, en la tercera se unen los grupos $\{7,8\}$ y $\{9,10\}$ y así sucesivamente. Obviamente el criterio de unión se obtiene a partir de la matriz de distancia, mediante procesos algorítmicos que pasamos a describir.

	1	2	3	4	5	6	7	8	9	10
1	,000	1,490	5,440	2,440	8,290	14,690	22,690	21,640	38,090	36,040
2	1,490	,000	1,250	4,250	8,000	9,000	25,000	21,250	41,000	36,250
3	5,440	1,250	,000	9,000	11,250	7,250	31,250	25,000	48,250	41,000
4	2,440	4,250	9,000	,000	2,250	10,250	10,250	10,000	21,250	20,000
5	8,290	8,000	11,250	2,250	,000	5,000	5,000	3,250	13,000	10,250
6	14,690	9,000	7,250	10,250	5,000	,000	16,000	9,250	26,000	18,250
7	22,690	25,000	31,250	10,250	5,000	16,000	,000	1,250	2,000	2,250
8	21,640	21,250	25,000	10,000	3,250	9,250	1,250	,000	4,250	2,000
9	38,090	41,000	48,250	21,250	13,000	26,000	2,000	4,250	,000	1,250
10	36,040	36,250	41,000	20,000	10,250	18,250	2,250	2,000	1,250	,000

Tabla 16.3: Ejemplo de matriz de proximidad, basada en distancias.

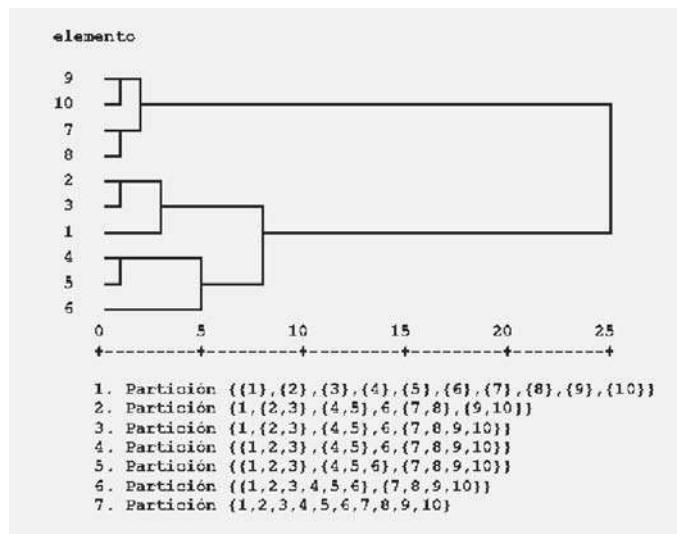


Figura 16.2: Ejemplo de agrupamiento por enlace completo.

16.4.2 Algoritmos para el agrupamiento jerárquico

El propio concepto de agrupamiento jerárquico nos indica que los algoritmos para obtenerlo serán de tipo aglomerativo, de forma que partiendo de una partición en la que cada elemento forma un grupo se van obteniendo nuevas particiones uniendo

grupos entre sí. Un primer enfoque para el desarrollo de estos algoritmos es el basado en grafos donde se considera que cada elemento es un vértice de un grafo y se van generando particiones, conectando los vértices de menor distancia. Aparecen entonces dos formas de agrupamiento [Jain y Dubes, 1988]:

- **Agrupamiento de enlace simple** (*Single-link clustering*). Los grupos se obtienen buscando las componentes conexas de grafo y se termina cuando todos los vértices están conectados.
- **Agrupamiento de enlace completo** (*Complete-link clustering*). Los grupos se obtienen buscando los subgrafos completamente conectados (cliqués), es decir, tendremos un grupo de dos vértices si hemos colocado una arista, tendremos un grupo de tres vértices si hemos colocado las tres aristas que los unen, etc. El algoritmo termina cuando hemos incluido todos los vértices en un grupo.

Un enfoque diferente y más general para obtener agrupamientos jerárquicos es debido a Johnson [Johnson, 1967] y se basa en sucesivas transformaciones de la matriz de proximidad, que para este algoritmo siempre es de distancia, reduciendo la dimensión de la misma siempre que se forme un nuevo grupo. La idea consiste en trabajar con una *matriz de distancia entre grupos*, y que ésta se vaya calculando iterativamente a partir de la matriz de la etapa anterior. La generalidad de este enfoque radica en el hecho de que la distancia entre grupos se puede calcular de distintas formas y que dependiendo de cómo se calcule dicha distancia aparecen diferentes formas de agrupamiento. Lance y Williams [Lance y Williams, 1967], han establecido la forma general de dicho cálculo y los parámetros que dan lugar a los diversos enfoques (veáse el Algoritmo 16.1).

Algoritmo 16.1 Algoritmo para el agrupamiento jerárquico.

Entradas: La matriz D de distancias de partida

Entradas: El conjunto de elementos $I = \{x_1, x_2, \dots, x_n\}$

- 1: $m = 0$
 - 2: $D_m = D$
 - 3: $L(m) = 0$
 - 4: $\mathcal{C}_m = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$ {Partición inicial con un elemento por grupo}
 - 5: **repetir**
 - 6: Sean $R, S \in \mathcal{C}_m$ tales que $D_m(R, S) = \min_{l, k \in \mathcal{C}_m, l \neq k} D_m(l, k)$ {Se localizan los dos grupos que estén más cercanos}
 - 7: $L(m + 1) = D_m(R, S)$
 - 8: $K = R \cup S$ {Formamos un nuevo grupo con los grupos más cercanos}
 - 9: $\mathcal{C}_{m+1} = \mathcal{C}_m \cup K - R - S$ {Se crea una nueva partición insertando el nuevo grupo y eliminando los que se han fusionado}
 - 10: Se elimina de D_m la fila y columna correspondiente a R o S y la fila y columna del grupo que queda se asigna a nuevo grupo K
 - 11: **para todo** $T \in \mathcal{C}_m$ tal que $T \neq K$ **hacer**
 - 12: $D_{m+1}(K, T) = a(R)D_m(R, T) + a(S)D_m(S, T) + bD_m(R, S) + c|D_m(R, T) - D_m(S, T)|$
 - 13: **fin para**
 - 14: $m = m + 1$
 - 15: **hasta** $|\mathcal{C}_m| = 1$ {Repetir hasta que todos los elementos estén agrupados en el mismo grupo}
-

METODO	a(R)	a(S)	b	c
Enlace Simple	1/2	1/2	0	-1/2
Enlace completo	1/2	1/2	0	1/2
Media de grupos	n_R/n_K	n_S/n_K	0	0
Centroide	$n_R/(n_R + n_T)$	$n_S/(n_S + n_T)$	$-(n_R n_S)/n_K^2$	0
Método de Ward	$(n_s + n_T)/(n_K + n_T)$	$(n_R + n_T)/(n_K + n_T)$	$-(n_T)/(n_K + n_T)$	0

Tabla 16.4: Coeficientes para el agrupamiento jerárquico.

En la Tabla 16.4 se pueden ver los valores que pueden tomar los coeficientes de la Expresión de línea 12 del anterior algoritmo para los algoritmos aglomerativos de agrupamiento jerárquico más conocidos (n_X indica el número de elementos que tiene el grupo X):

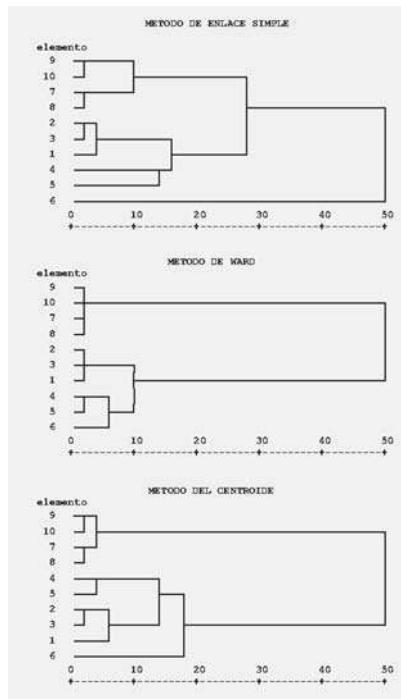


Figura 16.3: Ejemplos de distintos métodos de agrupamiento.

La elección de un método u otro depende de las propiedades que se busquen en los grupos, siendo el resultado muy dependiente del método, como se muestra en la Figura 16.3 donde se recogen agrupamientos según los métodos de enlace simple, del centroide y de Ward, aplicados al mismo ejemplo que se presentó en el apartado anterior, cuya matriz de distancia está representada en la Tabla 16.3 y cuyo agrupamiento por medio de enlace completo se muestra en la Figura 16.2.

En [Jain y Dubes, 1988] se puede encontrar un estudio teórico muy completo acerca de las propiedades del agrupamiento jerárquico, y en [Pang y otros, 2006] se recoge un interesante análisis acerca de las ventajas e inconvenientes de su uso. En el apartado 16.6 presentaremos algunas de las nuevas tendencias en agrupamiento jerárquico que pueden ser aplicadas a un gran volumen de datos.

16.5 Técnicas de agrupamiento particional

16.5.1 Ideas iniciales

El problema de agrupamiento particional puede formalizarse como sigue:

Dados n elementos representados en un espacio d -dimensional en el que hay definida una función de distancia, determinar una partición de los mismos en k subconjuntos o grupos, tales que los elementos incluidos en un grupo se parezcan más entre ellos de lo que se parecen a los clasificados en otros grupos.

El número k de grupos a generar puede estar definido previamente o no, aunque la práctica generalidad de los algoritmos actualmente en uso suponen que es un parámetro que debe estar fijado. A partir de ahora nos referiremos exclusivamente a estos algoritmos. Algoritmos con número de clases no fijado aparecen descritos en [Jain y Dubes, 1988], que contiene también un conjunto de citas al respecto. En cualquier caso lo que siempre hay que establecer es un criterio para medir tanto la coherencia de un grupo como la de un conjunto de grupos (agrupamiento). En este sentido podemos decir que:

- Existen métodos basados en “criterios globales” que suponen que cada grupo está representado por un prototipo, de modo que cada elemento se asigna al grupo cuyo prototipo esté más cercano. Se usan en este enfoque medidas de coherencia basadas en la distancia de cada elemento a su prototipo y dependiendo de la distancia que se considere aparecerán distintas medidas.
 - Para datos con atributos continuos, el prototipo de un grupo es habitualmente la media de los elementos que lo integran (*centroide*). El método de las k-medias que estudiaremos en el apartado 16.5.2 pertenece a esta categoría.
 - En el caso de atributos cualitativos se suele utilizar el elemento más representativo del grupo (*medoide*). Veremos algunas ideas acerca de estos métodos en la sección 16.6.
- Por el contrario, los métodos basados en “criterios locales” forman los grupos utilizando la estructura local de los datos, ejemplos de esta forma de trabajar son los métodos basados en la identificación de regiones de alta densidad de puntos o aquellos que asignan al mismo grupo un elemento y sus k -vecinos más cercanos. Uno de los métodos más conocidos dentro de este enfoque es DBSCAN [Ester y otros, 1996], que veremos con más detalle en la sección 16.5.3.

En [Jain y otros, 1999] puede encontrar un estudio bastante amplio de los *métodos de vecino más cercano* y en [Pang y otros, 2006] se pueden encontrar técnicas basadas en identificación de regiones. Un resumen de los enfoques más actuales de las técnicas de k -medoides se encuentra en [Berkhin, 2002], que también incluye una amplia bibliografía acerca del tema. Estos temas se tratarán en la sección 16.6.

16.5.2 El método de las k -medias

Tal y como ya hemos comentado el método de las k -medias es muy simple. Su aplicación requiere dos parámetros iniciales: el número de grupos, k , y sus respectivos centroides iniciales, que pueden ser elegidos por el usuario u obtenerse por medio de algún procesamiento previo. Una vez que disponemos de dichos datos, cada elemento es asignado al grupo del centroide más cercano, obteniendo de esta forma la composición inicial de los grupos. Una vez obtenidos estos grupos, se recalculan los centroides y se hace una nueva reasignación. El proceso se vuelve a repetir hasta que los centroides no cambian. El proceso completo queda detallado en el algoritmo 16.2.

Algoritmo 16.2 Algoritmo de las k -medias.

Entradas: El número de grupos, k , y sus respectivos centroides \mathbf{c}_i , $i \in \{1, \dots, k\}$
Entradas: La matriz de elementos $X = x_{ij}$ compuesta por n elementos $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, cada uno descrito por d atributos $\mathbf{x}_i = x_{i1}, \dots, x_{id}$
Entradas: Una función de proximidad p , que suponemos definida en términos de disimilaridad

- 1: Inicializamos los nuevos centroides, $\forall i \in \{1, \dots, k\}$, $\mathbf{c}_i^* = \mathbf{c}_i$
- 2: **repetir**
- 3: $\forall i \in 1, \dots, k, G_j = \emptyset$ {Se inicializan los grupos}
- 4: $\forall i \in 1, \dots, k, \mathbf{c}_i = \mathbf{c}_i^*$ {Se actualizan los centroides}
- 5: **para todo** $\mathbf{x}_i \in X$ **hacer**
- 6: Calcular j tal que $p(\mathbf{x}_i, \mathbf{c}_j) = \min_{l \in \{1, \dots, k\}} (p(\mathbf{x}_i, \mathbf{c}_l))$ {Localizar el centroide más próximo}
- 7: $G_j = G_j \cup \{\mathbf{x}_i\}$ {Se asigna \mathbf{x}_i al grupo del centroide más próximo}
- 8: **fin para**
- 9: **para** $j = 1$ hasta k **hacer**
- 10: $\mathbf{c}_j^* = \sum_{\mathbf{x}_i \in G_j} \mathbf{x}_i / |G_j|$ {Recalcular los nuevos centroides}
- 11: **fin para**
- 12: **hasta** $\forall i \in \{1, \dots, k\}$, $\mathbf{c}_i^* = \mathbf{c}_i$
- 13: **devolver** $\forall j \in \{1, \dots, k\}, G_j, \mathbf{c}_j$.

La versión original de este método se basaba en la utilización de la distancia euclídea, con lo que, y teniendo en cuenta que los centroides se definen como la media aritmética de los elementos de cada grupo, lo que se está minimizando en la expresión de la línea 6 del algoritmo 16.2 es la desviación con respecto a la media de los elementos del grupo, o lo que también se denomina la suma de los errores cuadráticos. No obstante se pueden considerar cualquier función de distancia o índice de semejanza, cambiando en este último caso, el criterio de minimización por el de maximización. De hecho se han utilizado con bastante frecuencia la distancia de Manhattan (véase la Tabla 16.1), considerando como centroide la mediana de los grupos, y la medida de similitud del coseno, (véase la Expresión 16.1).

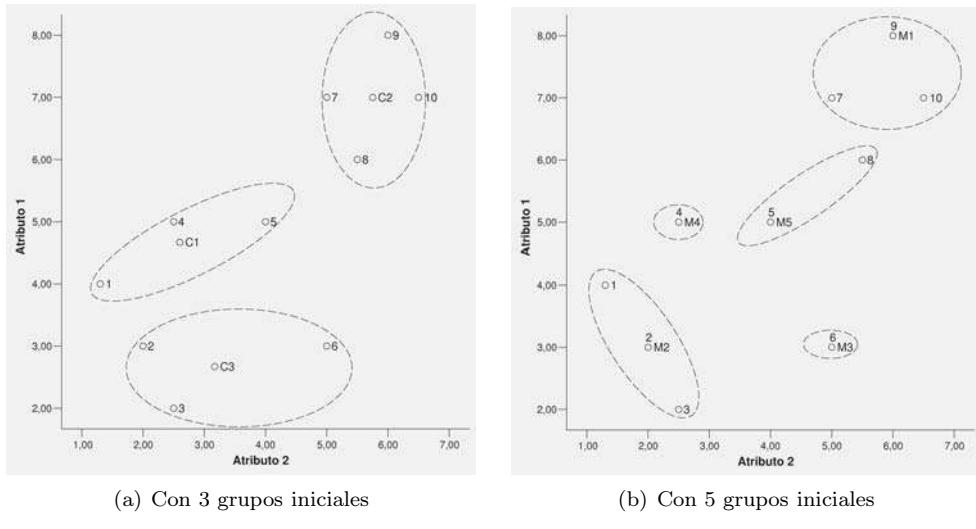


Figura 16.4: Ejemplo del método de las k -medias, con selección inicial de centroides aleatoria.

N. de grupos	Sin selección previa	Con selección previa
3	1.083	0.980
5	0.650	0.590

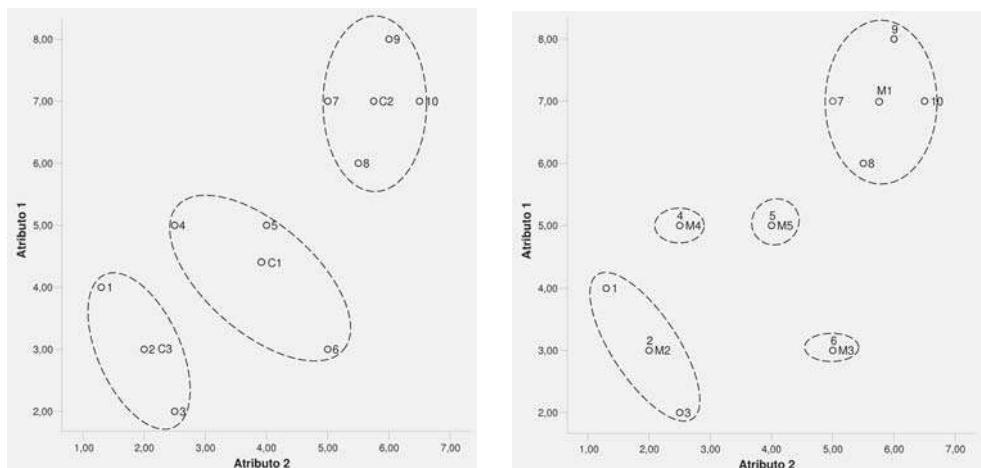
Tabla 16.5: Ejemplos de medida SSE en distintas aplicaciones del método de las k -medias.

Como ya se ha comentado con anterioridad, el método de las k -medias es probablemente el más popular y utilizado de todos los métodos particionales. No obstante, hay que tener en cuenta que es fuertemente dependiente de los parámetros de entrada, es decir, del número de grupos considerados y de los centroides que se eligen de partida. En las Figuras 16.4-(a) y 16.4-(b) se presentan dos ejemplos de agrupamiento de 10 elementos, definidos mediante dos atributos, cuya matriz de distancia se representa en la Tabla 16.3. En el primero se consideran 3 grupos, $k = 3$ y en el segundo 5, $k = 5$, y con selección incial de centrodies aleatoria. Como puede comprobarse hay una gran diferencia en los resultados.

Dada la gran dependencia de este método de sus parámetros iniciales, una buena medida de la bondad del agrupamiento es justamente la suma total de la proximidad que se intenta minimizar:

$$SSE = \sum_{j=1}^k \sum_{\mathbf{x} \in G_j} \frac{p(\mathbf{x}_i, \mathbf{c}_j)}{n} \quad (16.2)$$

En el caso de que trabajemos con la distancia euclídea, la Expresión 16.2 se corresponde con la del error cuadrático global. Obviamente este valor será tanto menor cuanto más grupos consideremos, pero esta opción puede no ser la más adecuada. Por ello, existen algunas técnicas de postprocesamiento [Pang y otros, 2006], que permiten mejorar el agrupamiento obtenido y técnicas de detección de “elementos extraños” (outliers). Tal y como se comenta en [Hang y Kamber, 2000], un procedimiento para fijar los parámetros de partida consiste en realizar un agrupamiento jerárquico previo y partir con alguno de los agrupamientos que proporcione. Por ejemplo, si al mismo conjunto de datos comentado anteriormente le aplicamos el método jerárquico de enlace completo (véase la Figura 16.2), un agrupamiento en tres grupos puede ser $P_1 = \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9, 10\}\}$. En cambio, si aplicamos el método de enlace simple 16.3), un agrupamiento en cinco grupos puede ser $P_2 = \{\{1, 2, 3\}, \{4\}, \{5\}, \{6\}, \{7, 8, 9, 10\}\}$. En las Figuras 16.5-(a) y 16.5-(b) se pueden ver los resultados obtenidos después de aplicar el método de las k -medias, tomando como centroides iniciales los de estos agrupamientos. Como puede verse, el algoritmo no ha modificado los grupos de partida, y los resultados son diferentes de los que se muestran en las Figuras 16.4-(a) y 16.4-(b), en los que la selección de los centroides se hizo de forma aleatoria. La Tabla 16.5 muestra los valores de SSE para cada caso; como puede verse, el valor disminuye con el número de grupos y la elección inicial usando un agrupamiento obtenido por un método jerárquico mejora esta medida de bondad.



(a) Con 3 grupos iniciales obtenidos mediante el método de enlace simple

(b) Con 5 grupos iniciales obtenidos mediante el método de enlace simple

Figura 16.5: Ejemplo del método de las k -medias con 3 (a) y 5 (b) grupos.

16.5.3 DBSCAN: un método basado en el análisis de densidad

Tal como se ha comentado, los métodos de agrupamiento basados en la densidad intentan encontrar regiones del espacio con una alta densidad de elementos, que estén separadas por otras regiones de baja densidad. Obviamente todos los métodos de esta categoría se basan en el concepto de “densidad” de una región, habiéndose desarrollado diversas formas de medirla, algunas de las cuales tienen fundamento estadístico. En este caso, se parte del conocimiento de la distribución de probabilidad conjunta entre elementos y grupos para obtener la distribución de probabilidad asociada a cada grupo y, a partir de ella, se desarrolla un procedimiento de asignación de elementos a los grupos. Los métodos basados en *Análisis Discriminante* y los de *Mezcla de Densidades* (véase [Guha y otros, 1998; Guha y otros, 2000]) pertenecen a esta categoría.

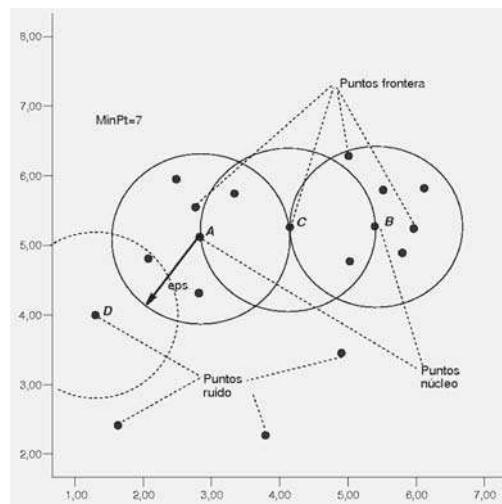


Figura 16.6: Conceptos básicos referentes a DBSCAN.

Uno de los algoritmos más interesantes y utilizados es DBSCAN, que si bien es bastante simple nos va a servir para ilustrar muchas de las ideas subyacentes en este enfoque. DBSCAN utiliza la denominada *densidad basada en centros*, la cual estima la densidad de cada punto (elemento) concreto, contando el número de puntos que caen dentro de un entorno centrado en él de radio eps . En la Figura 16.6, se presenta una distribución de puntos en un espacio bidimensional y varios entornos con radio eps centrados en algunos puntos. Podemos ver que la densidad en el punto A es de 7, mientras que la de D es de 2. Está claro que la densidad calculada de esta forma depende del radio que se tome. Si este es suficientemente grande, la densidad de cada punto será n , (el número total de puntos). Por el contrario, si es suficiente pequeño la densidad tendrá valor 1 en cualquier punto. Una vez que se ha fijado un valor, eps , para el radio y un número de puntos mínimo, MinPt , suficiente para considerar que un entorno tiene la densidad adecuada para formar un grupo, todo punto del espacio de elementos se puede clasificar como:

- **Punto Núcleo.** Aquél que es el centro centro de un entorno de radio eps que tiene más de MinPt puntos. En la Figura 16.6 son punto núcleo el A y el B . Se considera que pertenece al interior de un grupo.
- **Punto Frontera.** Aquél que se encuentra en un entorno de radio eps que tiene como centro un punto núcleo. En la Figura 16.6 se pueden observar varios puntos frontera. Puede ocurrir que un punto frontera pertenezca al entorno de varios puntos núcleo, como ocurre con el punto C en la Figura 16.6.
- **Punto ruido.** Aquél que no es núcleo, ni frontera. Se supone que va a estar en regiones muy poco densas y que no va a formar parte de ningún grupo. En la Figura 16.6 aparecen varios puntos frontera, entre ellos el D .

Teniendo en cuenta estos tipos de puntos, el ciclo básico de trabajo de DBSCAN es muy sencillo:

- Se colocan en un mismo grupo todos los puntos núcleo que distan entre sí menos de eps , aceptando un criterio de transitividad, es decir, si \mathbf{n}_1 y \mathbf{n}_2 son dos puntos núcleo, y la distancia entre ellos es menor que el radio, $f(\mathbf{n}_1, \mathbf{n}_2) \leq \text{eps}$, y tenemos que un tercer punto núcleo cumple que $f(\mathbf{n}_2, \mathbf{n}_3) \leq \text{eps}$, entonces $\mathbf{n}_1, \mathbf{n}_2$ y \mathbf{n}_3 pertenecen al mismo grupo. Se dice entonces que \mathbf{n}_1 y \mathbf{n}_2 (o \mathbf{n}_2 y \mathbf{n}_3) son “directamente densidad alcanzables” mientras que \mathbf{n}_1 y \mathbf{n}_3 se dice que son “densidad alcanzables”.
- También se asignan al mismo grupo todos los puntos frontera asociados a cada punto núcleo. Habrá que dar algún criterio para el caso en que un cierto punto pertenezca a los entornos de dos núcleos que no están en el mismo grupo.
- Se eliminan todos los puntos ruido.

DBSCAN funciona realmente de forma iterativa, ya que, fijados sus parámetros eps y MinPt , va considerando punto a punto, estableciendo su entorno de radio eps y viendo si es o no núcleo. En el caso de que lo sea, se construye un grupo con dicho entorno y se buscan otros núcleo que sean densidad alcanzables a partir de él. Si existe alguno, el grupo generado inicialmente se une a aquél al que pertenezca este núcleo. El proceso termina cuando ningún punto puede ser añadido a ningún grupo.

DBSCAN es un algoritmo potente y sencillo que puede optimizarse para espacios de baja dimensionalidad y que produce grupos complejos para los cuales no hay ninguna hipótesis de “centralidad” o “globularidad”, como en el caso del método de las k -medias. No obstante, depende fuertemente de los valores fijados para eps y MinPt . Este es un problema general de todos los métodos presentados, pero en el caso de DBSCAN el problema se agudiza, ya que cuando se tiene un gran volumen de datos y una alta dimensionalidad, puede ocurrir que haya zonas del espacio de elementos muy densas, con grupos que incluyan muchos puntos, y otras con una densidad más baja y con grupos menos densos. Si se toman los mismos parámetros para ambas zonas, los puntos de la segunda se considerarán como ruido, con lo que se pierde mucha información.

Existen generalizaciones de DBSCAN que permiten evitar en lo posible estos problemas. OPTICS [Akerst y otros, 1999] trabaja con regiones de densidad variable, considerando valores de eps menores. Un estudio detallado del OPTICS se puede encontrar en [Hang y Kamber, 2000]. Algunas consideraciones adicionales acerca de la complejidad de DBSCAN y sus problemas de uso se pueden encontrar en [Pang y otros, 2006]. Un estudio bastante completo de las extensiones de DBSCAN puede encontrarse en [Berkhin, 2002] y en [Hang y Kamber, 2000].

16.6 Nuevos resultados y extensiones

16.6.1 Resultados recientes sobre agrupamiento jerárquico

Las técnicas de agrupamiento jerárquico siempre han tenido el inconveniente de ser muy costosas desde el punto de vista computacional como consecuencia de operar con la matriz de proximidad completa. Este inconveniente se agrava cuando se trabaja con una gran cantidad de elementos, ya que, en su versión clásica, estas técnicas obtienen el conjunto de todos los posibles agrupamientos, desde el inicial con n grupos hasta el último con un sólo grupo. Las primeras versiones que tratan de evitar estos inconvenientes son los algoritmos llamados DIANA y AGNES [Kauffman y Rousseeuw, 1990]. El primero es de tipo divisivo y el segundo aglomerativo, pero en ambos se especifica el número de grupos que se desea como una condición de terminación, de modo que los algoritmos finalizan cuando se alcanza este número.

También se considera un gran inconveniente de los métodos jerárquicos clásicos, el hecho de que, cuando se toma la decisión de unir dos grupos (en los aglomerativos) o dividir uno (en los divisivos), no se puede volver atrás, ya que cada nuevo paso de las iteraciones parte del agrupamiento generado en el paso anterior. Este procedimiento tan rígido puede llevar a resultados erróneos, además de que la forma en que se toman las decisiones de unir o separar grupos implica evaluar un buen número de elementos o grupos y, por lo tanto, gran coste computacional. Un enfoque muy interesante que se ha seguido para resolver estos problemas, consiste en mejorar la calidad de los grupos obtenidos utilizando otras técnicas de agrupamiento, realizando de este modo un proceso de múltiples fases. A continuación, comentamos resumidamente los métodos más conocidos que siguen este enfoque.

- **BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies) [Zhang y otros, 1996]. Inicialmente particiona los elementos de forma jerárquica, utilizando estructuras de árboles, y posteriormente aplica otros algoritmos de agrupamiento para refinar el resultado. Este algoritmo se basa en el uso de resúmenes de los datos (estadísticos suficientes) para sustituir a los datos originales y es fácilmente escalable y muy usado en problemas de Minería de Datos. Se introducen dos conceptos:
 - El descriptor del grupo, CF (clustering feature). Para cada grupo con M elementos CF se define como la tupla:

$$CF = (M, LS, SS)$$

donde LS es la suma de las componentes del vector y SS la suma de los cuadrados de sus componentes. Un CF almacena un resumen estadístico de la información asociada a un grupo.

- El árbol de descriptores, **CF -tree**, que no es más que un árbol equilibrado cuyos nodos son CF y que se va construyendo conforme los grupos se van generando. El criterio de construcción de este árbol contempla dos parámetros: el factor de ramificación B , que especifica el número máximo de hijos que puede tener un nodo, y el umbral T que fija el diámetro máximo de los grupos almacenados en las hojas del árbol.

El algoritmo trabaja en dos fases. En la primera se explora la base de datos para construir un CF -tree inicial en memoria principal, que puede verse como una compresión multinivel de los datos y que trata de preservar la estructura de agrupamiento inherente a los datos. En la segunda se aplica un algoritmo de agrupamiento a las hojas del CF -tree.

BIRCH trata de producir los mejores grupos con los recursos computacionales disponibles. El análisis experimental del mismo ha probado la escabilidad lineal del algoritmo con respecto al número de elementos y la buena calidad de los resultados obtenidos. No obstante, dado que cada nodo de un CF -tree sólo puede almacenar un número limitado de elementos, es posible que no corresponda a un grupo real. Además, en el caso de que los grupos no sean esféricos, puede trabajar mal, ya que está basado en la noción de diámetro para controlar la frontera de los grupos.

- **CURE Clustering Using REpresentatives** [Guha y otros, 1998]. La mayoría de los algoritmos de agrupamiento favorecen la creación de grupos de forma esférica y se ven muy afectados por la existencia de elementos extremos (outliers). CURE, que integra algoritmos jerárquicos y particionales, intenta resolver estos inconvenientes.

Este método emplea un enfoque muy novedoso de agrupamiento jerárquico que se encuentra a mitad de camino entre los métodos de enlace ponderado y de enlace completo, ya que en lugar de utilizar un único punto, o todos ellos para representar un grupo, se elige un número fijo de puntos representativos. Estos puntos se generan eligiendo primeramente un conjunto de puntos bien distribuidos sobre el grupo, y posteriormente reduciendo la distancia de estos al centro del grupo un determinado factor, denominado *factor de reducción*. Los grupos con puntos representativos más cercanos se unen en cada paso del algoritmo.

Dado que usa más de un punto representativo por grupo, CURE se ajusta bien a la geometría de formas no esféricas y elimina el efecto de enlace simple. El proceso de reducción permite eliminar puntos extremos. El algoritmo trabaja con grandes bases de datos utilizando un proceso adicional de muestreo. La muestra obtenida se divide en un número de grupos mayor que el que se desea obtener. Con estos grupos se realiza la determinación de los puntos representativos, una reducción para eliminar extremos, y un proceso aglomerativo hasta obtener el número de grupos deseados.

CURE produce grupos de alta calidad en presencia de extremos y formas complejas de distintos tamaños, y es escalable a grandes bases de datos sin sacrificar la calidad de los grupos. Además, necesita relativamente pocos parámetros: tamaño de la muestra, número de grupos deseados, y factor de reducción, no siendo muy sensible a la variación de los mismos.

Otro algoritmo aglomerativo desarrollado por los mismos autores es **ROCK** [Guha y otros, 2000], orientado al manejo de atributos categóricos.

- **CHAMELEON** [Karypis y otros, 1999]. Explora un modelo dinámico de agrupamiento jerárquico. En su proceso de agrupamiento se unen dos grupos si su interconectividad y cercanía se corresponden con la conectividad interna y la cercanía de los elementos que están en ellos. El proceso de unión basado en un modelo dinámico, facilita el descubrimiento de grupos naturales y homogéneos, y se aplica a todo tipo de datos una vez que se ha establecido una adecuada función de similaridad.

Este algoritmo intenta evitar las debilidades de CURE y ROCK. CURE, al igual que otros métodos relacionados, ignora la información acerca de cómo están interconectados los elementos de dos grupos que se van a unir. ROCK, por su parte, se fija en la interconectividad de los elementos de los dos grupos, pero no hace uso de la distancia entre grupos como consecuencia del uso del método de enlace ponderado y otros esquemas relacionados.

CHAMELEON utiliza primero un algoritmo divisivo basado en grafos para agrupar los datos en un número relativamente grande de pequeños grupos. Después utiliza un algoritmo jerárquico aglomerativo para obtener los grupos finales. Para determinar la pareja de grupos más cercanos se usa tanto la interconectividad como la cercanía de los grupos, así como las características de cohesión internas de los grupos.

Para más información acerca de estos algoritmos, se puede consultar también [Berkhin, 2002] y [Hang y Kamber, 2000].

16.6.2 Extensiones a los métodos particionales prototípicos: los métodos de k-medoides

Como ya hemos comentado, el método de las k -medias es muy sensible a la presencia de elementos extraños, dado que un elemento con valores de atributos muy grandes puede distorsionar la distribución de los datos y el valor del centroide correspondiente. Por estas razones se ha propuesto una alternativa al uso del centroide en la que éste sustituye el prototipo de cada grupo por un punto del mismo considerado representativo. Este punto, que se denomina *medoide*, es el que está situado de forma más central en el grupo, desarrollándose todo el proceso iterativo del método de las k -medias y minimizando las sumas de las distancias de cada punto a los medoides considerados. Existen varios algoritmos basados en esta idea, algunos de los cuales pasamos a comentar.

- **PAM** [Kauffman y Rousseeuw, 1990]. Es uno de los algoritmos más antiguos y resulta bastante sencillo.
 - Se parte de una selección inicial de k medoides. Como en el caso de las k -medias, estos medoides se utilizan para generar una partición en k grupos del conjunto total de elementos.
 - Para cada grupo obtenido se intenta reemplazar el medoide asociado a él por algún punto del mismo grupo que sea más idóneo. Esto se hace considerando cada punto del grupo como candidato y calculando la distancia total del resto los elementos del grupo a dicho punto. Si hay mejora con respecto a la del medoide actual, este es sustituido por el punto en cuestión, en caso contrario se mantiene.
 - Se recalculan los grupos en base a los nuevos medoides seleccionados y se vuelve con ellos al paso anterior.
 - El proceso termina cuando no se cambian los medoides en una iteración.

Este método es más robusto que el de las k -medias en presencia de puntos extraños, pero el proceso es computacionalmente mucho más costoso que el de las k -medias. De hecho, si consideramos un conjunto de n elementos el coste computacional de cada iteración es de $O(k(n - k))^2$. En cualquier caso, sigue siendo muy dependiente del número de grupos y de la selección inicial de los prototipos.

- **CLARA** [Kauffman y Rousseeuw, 1990]. PAM trabaja bien para conjuntos de datos pequeños, pero no se adapta bien a grandes bases de datos. Para esta situación, la alternativa propuesta por sus diseñadores es CLARA, basado en un proceso de muestreo. La idea básica es que, si se toma una muestra suficientemente aleatoria y representativa, los medoides obtenidos con ella serán los mismos que los del conjunto total. Con esta idea, CLARA realiza sucesivos muestreos y aplica PAM a cada uno de ellos. Los conjuntos de medoides obtenidos se analizan sobre el conjunto total, seleccionando el que nos de menor distancia global. Con estos medoides se genera un nuevo proceso de muestreo y una nueva iteración. La complejidad de cada iteración es ahora de $O(kS^2 + k(n - k))$, donde S denota el tamaño de la muestra.

La efectividad de CLARA depende del tamaño de la muestra, si bien es fácilmente escalable y puede trabajar con grandes bases de datos. No obstante, hay que destacar que CLARA no obtiene un conjunto óptimo de medoides, ya que estos se extraen de datos muestreados y puede ocurrir que un punto óptimo no se considere nunca. En este sentido es muy importante la aleatoriedad del proceso de muestreo.

- **CLARANS** [Ng y Han, 1994]. Para mejorar la calidad y escalabilidad de CLARA se ha propuesto un nuevo algoritmo, denominado CLARANS, desarrollado en el contexto de las bases de datos espaciales. CLARANS es también de tipo k -medoide y combina técnicas de muestreo con PAM. La diferencia con CLARA

radica en que el proceso de muestreo no se hace para cada conjunto de medoides, sino que se realiza cada vez que se calcula uno de ellos. La idea consiste en considerar la búsqueda de los medoides óptimos como un proceso de búsqueda en un árbol donde cada nodo es un conjunto de k medoides. Para cada nodo concreto, con su conjunto de prototipos y su agrupamiento asociado, se obtiene un nuevo agrupamiento, denominado entorno, reemplazando un medoide del mismo por algún otro punto. El proceso prueba una serie de entornos generando puntos aleatoriamente. Si encuentra un entorno mejor que el agrupamiento considerado, el algoritmo se mueve a este nodo y comienza de nuevo a probar. En caso contrario se considera que se ha llegado a un óptimo local. Cuando se llega a un óptimo local, el algoritmo comienza con un nuevo conjunto de nodos obtenidos por medio de un muestreo aleatorio y una aplicación de PAM. El algoritmo termina cuando se han alcanzado un número suficiente de mínimos locales (datos experimentales recomiendan tomar dicho número igual a 2). La complejidad de CLARANS es de $O(n^2)$. En [Ester y otros, 1994] se presenta una mejora de este algoritmo obtenida mediante el uso de R^* -árboles.

16.7 Técnicas de agrupamiento borroso

En todos los métodos de agrupamiento presentados hasta ahora se ha supuesto, al menos implícitamente, la hipótesis de que el agrupamiento es exclusivo, en el sentido de que un elemento se asigna a un único grupo. En otras palabras, los elementos se partitionan en conjuntos disjuntos. Naturalmente, si los grupos son compactos y están bien separados esta es la mejor opción. Pero el problema aparece cuando los grupos tienen puntos comunes e incluso se solapan, en cuyo caso las fronteras de cada grupo no están definidas y existen puntos que pueden pertenecer a más de un grupo. Por lo tanto, aparecen grupos cuyas fronteras están mal definidas o “borrosas”. La teoría de Conjuntos Borrosos (*fuzzy sets*, véase el capítulo 7) desarrollada en la década de los 60 permite que un elemento pertenezca a un grupo con un cierto “grado de pertenencia” definido sobre el intervalo $[0,1]$. Para grupos ordinarios (grupos “crisp”), el grado de pertenencia para un punto solo puede ser 1 ó 0, según el punto esté o no en el grupo. Evidentemente, se trata de un caso particular de grupo borroso.

En estas condiciones cada grupo borroso $C_j; j \in \{1, \dots, K\}$ tiene asociada una función de pertenencia, μ_j :

$$\mu_j : X \longrightarrow [0, 1]$$

siendo $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ el espacio de elementos. Por lo tanto, el valor $\mu_{ij} = \mu_j(\mathbf{x}_i)$ mide el grado de pertenencia del elemento \mathbf{x}_i al grupo C_j , y será mayor cuanto mayor sea la confianza de que \mathbf{x}_i pertenezca a C_j . Si estuviéramos totalmente seguros de dicha pertenencia, entonces $\mu_{ij} = 1$. Los valores μ_{ij} constituyen lo que se denomina la *matriz de pertenencia*, que denominaremos U .

Aunque es bastante habitual imponer la condición (de partición borrosa):

$$\sum_{j=1}^k \mu_{ij} = 1, \forall i \in \{1, \dots, n\},$$

hay que hacer notar que el grado de pertenencia no tiene el mismo sentido que una probabilidad. Bajo una hipótesis probabilística, un punto pertenece solamente a un grupo que se determina por medio de un experimento aleatorio. Con una hipótesis basada en la lógica borrosa, un punto puede pertenecer a dos grupos a la vez, pudiéndose interpretar el grado de pertenencia, μ_{ij} , como el grado de compatibilidad del punto \mathbf{x}_i con el grupo C_j , entendido este como el resultado de una propiedad (o un conjunto de propiedades) expresadas de forma imprecisa. Este enfoque es muy útil cuando se busca una interpretación de los grupos, ya que, en muchos casos, las descripciones de los grupos obtenidos serán de tipo impreciso por serlo las etiquetas que los caracterizan. Este sería el caso, por ejemplo, si intentásemos agrupar un conjunto de coches en aquellos que son de gama “alta”, “media” o “utilitarios”, o si intentásemos agrupar un conjunto de parcelas atendiendo a las prácticas de cultivo que realizan sobre ellas.

Los problemas de agrupamiento han sido ampliamente tratados por medio de conjuntos borrosos desde 1966. Existen buenos libros sobre el tema, de entre los que destacaremos [Bezdeck y otros, 2005] por ser el más reciente y porque además contiene una amplia bibliografía.

16.7.1 Agrupamientos borrosos particionales

Casi todos los algoritmos de agrupamiento basados en conjuntos borrosos hacen uso del concepto de partición borrosa, lo que permite considerarlos como particionales con fronteras mal definidas entre grupos. No obstante, y como veremos posteriormente, esta teoría está estrechamente relacionada con el agrupamiento jerárquico. Existen distintos enfoques para diseñar algoritmos particionales de tipo borroso, la mayor parte de los cuales son generalizaciones más o menos directas del método de las k -medias. Una forma genérica de describir estos algoritmos es la siguiente.

1. Seleccionar una partición borrosa inicial de n elementos en k grupos definiendo la correspondiente matriz de pertenencia U .
2. Calcular los “centros” de los grupos borrosos asociados a U mediante la expresión:

$$c_j = \sum_{i=1}^n \mu_{ij} \mathbf{x}_i$$

3. Utilizando U , calcular el valor óptimo de una función objetivo, que habitualmente será una ponderación de alguna forma de error cuadrático, y que puede ser expresada de la siguiente forma:

$$E^2(U) = \sum_{i=1}^n \sum_{j=1}^k \mu_{ij} \|\mathbf{x}_i - \mathbf{c}_j\|^2 \quad (16.3)$$

4. En este punto debemos resolver un problema de optimización sobre los valores de pertenencia μ_{ij} , que estarán sujetos a la restricción de partición borrosa comentada anteriormente:

$$\sum_{j=1}^k \mu_{ij} = 1; \mu_{ij} \geq 0 \quad (16.4)$$

o bien a condiciones algo más suaves como:

$$\max_{j \in \{1..k\}} \mu_{ij} = 1; \mu_{ij} \geq 0 \quad (16.5)$$

5. Repetir desde el paso 2 hasta que los valores de U no cambien significativamente.

Las diferencias entre las distintas variantes de este proceso general se encuentran, de una parte, en la forma de la función objetivo 16.3, y de otra, en las distintas formas de calcular el óptimo de la misma.

El algoritmo particional borroso más popular es el método conocido como “fuzzy c-means”, que es una adaptación directa del algoritmo k -medias siguiendo el esquema anterior (véase [Bezdeck y otros, 2005]). Algunas variantes del mismo se pueden encontrar en [Dumitrescu y otros, 2000]. A título de ejemplo comentaremos que algunas variaciones se obtienen considerando:

1. Que el centro de un grupo borroso no es su media sino su valor más representativo, es decir:

$$\forall j \in \{1, \dots, k\}; c_j = \mathbf{x}_j \mid \mu_{ij} = \max_{i \in \{1, \dots, n\}} (\mu_{ij})$$

2. Otras funciones de distancia más generales que la que se describe en la Expresión 16.3. Por ejemplo, dada una partición borrosa definida por la matriz U , puede definirse la distancia entre dos puntos asociada al grupo C_j como:

$$\forall \mathbf{x}, \mathbf{y} \in X, d_j(\mathbf{x}, \mathbf{y}) = \min(\mu_j(\mathbf{x}), \mu_j(\mathbf{y})) \|\mathbf{y} - \mathbf{x}\|^2$$

En estas condiciones la distancia introducida en la Expresión 16.3 se transforma en:

$$S^2(U) = \sum_{i=1}^n \sum_{j=1}^k d_j(\mathbf{x}_i, \mathbf{c}_j) \quad (16.6)$$

y la forma de dicha expresión depende de qué valor tome $\mu_j(\mathbf{c}_j)$. En el caso de que consideremos $\mu_j(\mathbf{c}_j) = 1$ obtendríamos la Expresión 16.3.

Otras variantes del algoritmo utilizan una función de distancia adaptativa que depende de cada iteración, ya que utiliza el “diámetro” de cada clase en la expresión de la distancia. Este algoritmo puede ser analizado en profundidad en [Dumitrescu y otros, 2000]. Otra referencia importante dentro de este apartado es [Bezdeck y otros, 2005]. Ambos libros contienen a su vez conjuntos de referencias muy amplios acerca de este tema.

Existen otros métodos de agrupamiento borroso que pueden aplicarse cuando lo que se conoce no es la relación entre los elementos, sino que se tiene una cierta medida de la conexión entre los elementos y sus clases. Esta situación es similar a la que, en el caso de agrupamiento particional no borroso, se posee información acerca de la distribución de probabilidad de cada grupo. Es decir, de la probabilidad de que un determinado elemento pertenezca a una clase. Como comentamos en el apartado que ha dado origen a los métodos de agrupamiento “crisp” de tipo estadístico (Análisis Discriminante, Máxima Verosimilitud, etc.).

En el caso de los métodos de agrupamiento basados en conjuntos borrosos la generalización se hace en dos sentidos:

1. Considerando que los grupos son borrosos.
2. Considerando que la asociación entre los elementos y sus grupos no está dada por una distribución de probabilidad conjunta, sino por medio de un concepto más general que es el de una “Asignación Básica de Probabilidad”, es decir una “Evidencia” según la Teoría de Dempster-Shaffer.

El modelo general contiene los siguientes elementos:

1. $X = \{x_1, \dots, x_n\}$ conjunto finito de elementos a agrupar.
2. $C = \{C_1, \dots, C_k\}$ conjunto de grupos de X . Supondremos que $\{C_i\}$ son subconjuntos borrosos de X .
3. Una “asignación básica de probabilidad” (BAP) m sobre $\mathcal{P}(X \times C)$, representando la información disponible acerca de la relación elemento/grupo. m mide la probabilidad de que ciertos conjuntos de elementos y de grupos aparezcan conjuntamente.
4. Una función que mida el coste de “mal agrupamiento”, es decir, que indique el coste de considerar que un elemento pertenece a un grupo cuando, en realidad, pertenece a otro.

$$r : C \times C \rightarrow R^+ \cup \{0\}$$

r puede representarse como una matriz $k \times k$, $[r_{ij}]$.

En estas condiciones, lo que se busca es una familia de funciones de pertenencia $\Upsilon = \{\mu_1, \dots, \mu_n\}$ para los grupos, de manera que se haga lo menor posible alguna función de pérdida, por ejemplo:

$$R_i(\Upsilon) = \sum_{A \subset X} m(A/i) \sum_{l=1}^k r_{li} \sup_{\mathbf{x} \in A} \mu_l(\mathbf{x})$$

o bien,

$$T_i(u) = \sum_{A \subset X} m(A/i) \sum_{l=1}^k r_{li} \inf_{\mathbf{x} \in A} u_l(\mathbf{x})$$

donde $m(A/i)$ son medidas condicionadas. También se pueden limitar los posibles valores de las funciones de pertenencia introduciendo restricciones como las indicadas en las Expresiones 16.4 y 16.5. Los resultados correspondientes a estos modelos se pueden encontrar en [Delgado y otros, 1999]. Algunos casos particulares se proponen en [Vila y Delgado, 1983] y [Delgado y otros, 1997].

16.7.2 Agrupamientos jerárquicos y conjuntos borrosos

La Teoría de Conjuntos Borrosos ha resultado ser una herramienta muy adecuada para ser usada en problemas de clasificación y particularmente para generar potentes métodos de agrupamiento. Ahora bien, en el caso jerárquico la relación va más allá de la mera consideración de herramienta, una relación como consecuencia de la identidad entre un agrupamiento jerárquico con el concepto de relación de similitud borrosa.

De la misma forma que el concepto de conjunto borroso amplía el de conjunto clásico, el concepto de relación borrosa amplía el de relación clásica, concretamente el de relación de similitud amplía el de relación de equivalencia. En la sección 7.7 del capítulo 7 se puede ver la definición formal del relación borrosa, y en la sección 7.7.1 del mismo capítulo podemos ver la definición de relación de similitud.

Supongamos que tenemos una relación de similitud R definida sobre un conjunto X mediante la función de pertenencia μ_R :

$$\mu_R : X \times X \rightarrow [0, 1] \quad (16.7)$$

La importancia de este tipo de relaciones en las técnicas de agrupamiento borroso radica en el cumplimiento de las siguientes propiedades:

1. Sea $\alpha \in [0, 1]$ y sea la relación no borrosa $R_\alpha = \{(x, y) | \mu_R(x, y) \geq \alpha\}$, si μ_R es de similitud, entonces R_α es de equivalencia, es decir, induce una partición en el conjunto X .
2. Sea $\alpha, \beta \in [0, 1]$ tales que $\alpha \leq \beta$, y sean \mathcal{P}_α y \mathcal{P}_β las particiones inducidas en X por R_α y R_β respectivamente, se verifica que:

$$\forall B \in \mathcal{P}_\beta \exists A \in \mathcal{P}_\alpha \text{ tal que } B \subseteq A$$

es decir si $\alpha \leq \beta$, entonces \mathcal{P}_α y \mathcal{P}_β son “particiones encajadas”.

De estas dos propiedades se deduce fácilmente que los conceptos de agrupamiento jerárquico y relación de similitud borrosa son equivalentes, de forma que de todo agrupamiento jerárquico se puede obtener una relación de similitud borrosa, y de toda relación de similitud borrosa se puede obtener un agrupamiento jerárquico. Esta identificación de conceptos permite utilizar las medidas y propiedades asociadas a conjuntos borroso en criterios de búsqueda de particiones en agrupamientos jerárquicos. En [Delgado y otros, 1996], [Dumitrescu y otros, 2000] o [Bezdeck y otros, 2005] puede encontrarse explicaciones detalladas sobre el tema, así como acerca de la relación de los agrupamientos jerárquicos borrosos, con jerarquías de sistemas de control.

16.8 Herramientas software para el agrupamiento

Terminamos este capítulo comentando algunos de los paquetes software que ofrecen implementaciones de técnicas de agrupamiento. Obviamente no vamos a recoger todos los existentes ya que, debido a que estas técnicas actualmente se integran dentro de la Minería de Datos, todo nuevo paquete software de este tipo suele incluir algún módulo de agrupamiento. Por ello, en este apartado nos limitaremos a comentar algunas de las herramientas más conocidas, remitiendo a lector al portal **kdnuggets**¹, para más información al respecto.

Como comentamos en la introducción a este tema, el agrupamiento tiene una amplia historia. Inicialmente se consideró como un problema de análisis de datos y, por lo tanto, como un problema de tipo estadístico, posteriormente se incluyó como una técnica más dentro de la Minería de Datos. Por esta razón, las herramientas software que incluyen técnicas de agrupamiento se encuentran dentro de estos dos grupos, dentro de los cuales además, existen paquetes comerciales y de software libre. Nosotros vamos a comentar un ejemplo de cada uno de estos tipos.

Herramientas de Software Estadístico

- **SPSS.** Si bien SPSS² es uno de los paquetes comerciales más conocido dentro del ámbito estadístico, cabe también destacar las herramientas que oferta para realizar agrupamientos. Estas permiten realizar agrupamientos tanto desde el punto de vista más clásico, como algunas versiones más avanzadas. Estas herramientas son además bastante completas desde el punto de vista, funciones de distancia que contemplan o de tipos de atributos que se consideran.
 - El agrupamiento jerárquico (“conglomerados jerárquicos”, en el paquete) contempla los enfoques más conocidos: enlace simple, completo y medio, centroide, método de Ward, etc. También contempla las tres clases más habituales de atributo: binarios, categóricos y definidos sobre intervalos. Con respecto a medidas de distancia, además de la euclídea, oferta las distancias del coseno, correlación de Pearson, Minkowsky, etc. Oferta también

¹www.kdnuggets.com/software/index.html

²www.spss.com

medidas de distancia obtenidas a partir de los índices de semejanza más conocidos: Jacquard, Lance y William, etc.

- El método de las *k*-medias, (“conglomerados de K-medias” en el paquete) permite elegir el número de grupos que se desee, así mismo se pueden almacenar el conjunto de centros obtenidos, y se puede partir de un conjunto de centros previo que esté almacenado en un fichero.
 - El método que el paquete denomina “conglomerados en dos fases” ofrece una implementación de BIRCH (véase la sección 16.6).
- **R.** Tal como se describe en su página web inicio³ R es un lenguaje y un entorno de trabajo para el cálculo estadístico y desarrollos gráficos. Puede verse como una combinación de paquete estadístico y lenguaje de programación, en este sentido tiene una cierta analogía con Matlab. R es software libre desarrollado bajo GNU, trabaja bajo Windows, MacOS, Linux y distintas variantes de Unix, no está soportado por ninguna empresa comercial, pero tiene una comunidad de desarrolladores muy activa. Por esta razón, existe una gran cantidad de paquetes de las más variadas técnicas de Análisis de Datos. Los paquetes estadísticos básicos de R se pueden encontrar en la página web del proyecto R, los más avanzados, desarrollados como software libre por una amplia variedad de especialistas se encuentran en el portal *R-Project*⁴. Prácticamente cualquier técnica de análisis de datos, medianamente conocida, puede encontrarse en la lista que ofrece.

Con respecto al agrupamiento, aparte de técnicas clásicas como el método de las *k*-medias, en el paquete *cba*, se pueden encontrar implementaciones de ROCK, en el paquete *cclust*, AGNES y DIANA (véase la sección 16.6), PAM y CLARA (véase la sección 16.6.2) todos ellos en el paquete *cluster*. Existen también diversos paquetes de agrupamiento jerárquico y de validación de agrupamientos. También hay técnicas para tipos de datos especiales como son los provenientes de imágenes, datos espaciales o de tipo genómico (microarrays). Recomendamos al lector interesado visitar las páginas web asociadas.

- **S-PLUS.** S-PLUS pertenece a la misma familia que R, de hecho ambos son variantes de un mismo lenguaje inicial S, que tiene tres implementaciones: una antigua versión de S (S versión 3, S-PLUS 3.x y 4.x), la nueva implementación de S-PLUS 5.x y posteriores, y R. S-PLUS es una versión comercial de S desarrollada por Insightful Corporation⁵. Tiene un nivel de desarrollo científico similar al de R; pero con las facilidades de uso que proporciona un paquete comercial interactivo. No obstante hay que recordar que S-PLUS es también un lenguaje que hay que programar.

³<http://www.r-project.org>

⁴<http://CRAN.R-Project.org>

⁵<http://www.insightful.com>, en España lo comercializa Addlink (<http://www.addlink.es>)

Herramientas de Software de Minería de Datos

- **CLEMENTINE.** Aunque CLEMENTINE⁶ es una herramienta para la Minería de Datos (MD) desarrollada por SPSS, tiene un enfoque muy distinto de este paquete estadístico. Posee un lenguaje de tipo gráfico muy intuitivo y permite ir almacenando resultados intermedios que se usan en procesos posteriores. En este sentido resulta una herramienta de análisis muy cómoda. No obstante, desde el punto de vista de la MD, la oferta de posibles técnicas es bastante pobre. Concretamente desde el punto de vista del agrupamiento oferta el método de las *k*-medias, el método de Kohonen, basado en redes neuronales, y el método de las dos fases, es decir BIRCH que ya se oferta también en SPSS, no hay agrupamiento jerárquico, ni métodos de medoides.
- **WEKA.** Weka es un paquete software para MD desarrollado por la universidad de Waikato en Nueva Zelanda⁷. Es quizás el paquete de MD de software libre más popular. Tiene un lenguaje de comandos y un interfaz de uso muy intuitivo. Quizás su peor inconveniente sea la forma en que se le deben dar los datos, ficheros tipo textos donde se incluyen conjuntamente datos y atributos, si bien también permite acceso a la bases de datos. Desde el punto de vista de las técnicas, la mejores ofertas de Weka están en el ámbito de la clasificación no del agrupamiento, tiene un paquete de agrupamiento que incluye el método de las *k*-medias y alguna variante suya, y una implementación de EM, una técnica de análisis de densidad basada en el cálculo de una mixtura de funciones de densidad de probabilidad. También incluye una técnica de validación basada el análisis estadístico de las densidades de los agrupamientos.

16.9 Lecturas recomendadas

- Para un estudio en profundidad del problema del agrupamiento y sus técnicas más clásicas, el libro de Jain y Dubes ([Jain y Dubes, 1988]) es excelente. También presenta muy bien otros problemas que el agrupamiento conlleva tales como: preparación de datos, medidas de distancia, semejanza, etc., y validación de resultados. Este libro es un referente en el estudio de agrupamiento y se puede obtener una versión libre del mismo en la página de uno de los autores⁸.
- En general no existe un libro concreto que estudie todas las técnicas y enfoques con suficiente detalle. Por ello, para conocer a fondo una técnica concreta lo mejor es utilizar el trabajo donde se presenta. Sin embargo, existen varios libros de MD que las comentan de manera bastante amplia, el libro de Pang y Kumar ([Pang y otros, 2006]) es muy completo. La página web del mismo⁹ contiene mucho material útil, incluyendo un capítulo introductorio de agrupamiento.

⁶<http://www.spss.com/clementine>

⁷<http://www.cs.waikato.ac.nz/ml/weka/>

⁸http://www.cse.msu.edu/jain/Clustering_Jain_Dubes.pdf

⁹<http://www-users.cs.umn.edu/kumar/dmbook>

- El informe de P. Berkhin [Berkhin, 2002]¹⁰ contiene un interesante resumen acerca de agrupamiento y MD.
- Por último, una referencia muy completa acerca de agrupamiento difuso se puede encontrar en el libro [Bezdeck y otros, 2005].

16.10 Resumen

Terminamos este capítulo, insistiendo en las mismas ideas que se expusieron en su introducción. El problema del agrupamiento es tan amplio y aparece en tan distintas áreas que, incluso dedicando un libro completo a su estudio, sería imposible recoger todos los modelos, técnicas y algoritmos a que ha dado lugar. De hecho siguen desarrollándose modelos y algoritmos, especialmente para los nuevos tipos de problemas que han ido surgiendo dentro del análisis de datos, como son los de Minería de Textos y Minería Web (Web y Text Mining), o de Genómica.

En este sentido lo que se ha presentado en este capítulo es una introducción a los problemas y modelos, intentando sobre todo dar ideas claras acerca de los conceptos básicos. El lector interesado en el tema puede continuar su formación a partir de las lecturas y actividades recomendadas que pasamos a comentar.

16.11 Ejercicios propuestos

Si bien, algunos de los libros antes citados contienen amplias colecciones de ejercicios, (véase [Pang y otros, 2006]), entendemos que en un nivel de postgrado, las actividades prácticas deberían centrarse en el uso de algunas herramientas software y en la comparación de distintas técnicas frente a diversos tipos de problemas. En este sentido proponemos al lector que, utilizando algunos de los paquetes software mencionados en el apartado 16.8, resuelva algunos problemas de agrupamiento concretos. Puede encontrar un buen repositorio de datos en el **UCI Machine Learning Repository**¹¹.

¹⁰<http://citeseer.ist.psu.edu/berkhin02survey.html>

¹¹<http://www.ics.uci.edu/~mlearn/MLSummary.html>

Referencias

- AKERST, M.; BREUNING, M.; KRIESEL, H.P. y SANDERS, J.: «OPTICS: Ordering points to identify clustering structures». En: *Proc. of ACM-SIGMOD Int. Conf.*, pp. 49–60. Philadelphia USA 1999, 1999.
- BERKHIN, P.: «Survey Of Clustering Data Mining Techniques». *Informe técnico*, Accrue Software, San Jose, CA, 2002.
citeseer.ist.psu.edu/berkhin02survey.html
- BEZDECK, J.; KELLER, J.; KRISNAPURAM, R. y PAL, N.: *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Springer Verlag, 2005.
- DELGADO, M.; GOMEZ-SKARMETA, A.F. y VILA, M.A.: «On the use of Hierarchical Clustering in Fuzzy Modelling». *International Journal on Approximate Reasoning*, 1996, **14**, pp. 237–257.
- DELGADO, M.; GOMEZ-SKARMETA, A.F. y VILA, M.A.: «On the use of probability and possibility measures in fuzzy clustering». En: *Proc. of Fuzz-IEEE 97*, pp. 143–148. IEEE Pub., 1997.
- DELGADO, M.; GOMEZ-SKARMETA, A.F. y VILA, M.A.: «Pattern Recognition with Evidential Knowledge». *International Journal of Intelligent Systems*, 1999, **14(2)**, pp. 145–164.
- DUMITRIESCU, D.; LAZZERINI, B. y JAIN, L.C.: *Fuzzy sets and their applications to clustering and training*. CRC Press, Boca Raton, London, New York, Washington D.C., 2000.
- DURAN, B.S. y ODELL, P.L.: *Cluster Analysis: a survey*. Springer Verlag, Berlin, Heidelberg, New York, 1974.
- ESTER, M.; KRIESEL, H.-P.; SANDER, J. y X.XU: «A density-based algorithm for discovering clusters in large spatial databases». En: *Proc. 2nd. Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pp. 67–82. Oregon Usa August 1996, 1996.
- ESTER, M.; KRIESEL, H.-P. y X.XU: «A database interface for clustering in large spatial databases». En: *Proc. 1st. ACM SIGKDD*, pp. 94–99. Montreal Canada, 1994.
- GUHA, S.; RASTOGI, R. y SHIM, K.: «CURE: an efficient clustering algorithm for large databases». En: *1998 ACM-SIGMOD Int. Conf. Management of Data proceedings*, pp. 73–84. Sydney, Australia March 1998, 1998.
citeseer.ist.psu.edu/guha98cure.html
- GUHA, S.; RASTOGI, R. y SHIM, K.: «ROCK: A Robust Clustering Algorithm for Categorical Attributes». *Information Systems*, 2000, **25(5)**, pp. 345–366.
citeseer.ist.psu.edu/article/guha99rock.html

- HANG, J. y KAMBER, M.: *Data Mining: Concepts and Techniques*. Morgan Kauffman, 2000.
- JAIN, A. K.; MURTY, M.N. y FLYNN, P.J.: «Data Clustering : A Review». *ACM Computing Survey*, 1999, **31**, pp. 264–323.
- JAIN, K. y DUBES, R. C.: *Algorithms for Clustering Data*. Prentice Hall, New Jersey, 1988.
- JOHNSON, S. C.: «Hierarchical Clustering Schemes». *Psychometrika*, 1967, **2**, pp. 241–254.
- KARYPIS, G.; E.-H.HAN y KUMAR, V.: «CHAMELEON: Hierarchical Clustering Using Dynamic Modeling». *Computer*, 1999, **32(8)**, pp. 68–75.
citeseer.ist.psu.edu/karypis99chameleon.html
- KAUFFMAN, L. y ROUSSEEUW, P.J.: *Finding Group in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, New York, 1990.
- LANCE, G. N. y WILLIAMS, W.T.: «A general theory of classificatory sorting strategies: II. Clustering systems». *Computer Journal*, 1967, **10**, pp. 271–277.
- NG, R. T. y HAN, J.: «Efficient and Effective Clustering Methods for Spatial Data Mining». En: Jorgeesh Bocca; Matthias Jarke y Carlo Zaniolo (Eds.), *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pp. 144–155. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1994.
citeseer.ist.psu.edu/ng94efficient.html
- PANG, N. T.; M., STEINBACH y KUMAR, V.: *Introduction to Data Mining*. Addison Wesley, New Jersey, 2006.
- VILA, M.A. y DELGADO, M.: «Problems of Classification in a Fuzzy Environment». *Fuzzy Sets and Systems*, 1983, **9(83)**, pp. 229–239.
- WILSON, D. R. y MARTINEZ, T. R.: «Improved heterogeneous distance functions». *Journal of Artificial Intelligence Research*, 1997, **6**, pp. 1–34.
- ZHANG, T.; RAMAKRISHNAN, R. y LIVNY, M.: «BIRCH: an efficient data clustering method for very large databases». En: *1996 ACM-SIGMOD Int. Conf. Management of Data proceedings*, pp. 103–114. Montreal Canada June 1996, 1996.
citeseer.ist.psu.edu/zhang96birch.html

Capítulo 17

Aprendizaje de árboles y reglas de decisión

César Ferri Ramírez y María José Ramírez Quintana

Universidad Politécnica de Valencia

17.1 Introducción

Entre las distintas aproximaciones al aprendizaje supervisado, los llamados métodos basados en modelos se caracterizan por representar el conocimiento aprendido (el modelo, a veces también llamado hipótesis) en algún lenguaje de representación más rico que el utilizado para representar la evidencia a partir de la cual se hace el aprendizaje. Por lo tanto, estos métodos construyen generalizaciones explícitas de los casos de entrenamiento, de tal forma que luego se pueden aplicar directamente para clasificar otros casos no vistos. Por contra, los métodos basados en instancias representan el conocimiento aprendido como un conjunto de prototipos descritos en el mismo lenguaje usado para representar la evidencia. El prototipo puede ser uno de los ejemplos o bien puede obtenerse a partir de uno o más ejemplos (por ejemplo a partir de algún tipo de media). Una de las ventajas de los métodos basados en modelos frente a los basados en instancias es que una vez construido el modelo ya no es necesario mantener los casos de entrenamiento para clasificar nuevos ejemplos. Otra de las ventajas es que los modelos pueden obviar algunos de los atributos si no son relevantes para la clasificación. En [Quinlan, 1993] se analizan algunas de las ventajas del aprendizaje de modelos. Ya hemos visto en los temas anteriores algunos métodos basados en modelos como las redes neuronales o las máquinas de vectores soporte.

Sin embargo, para algunas áreas de aplicación no sólo es interesante disponer de un método que genere un modelo predictivo, sino que éste sea inteligible para el usuario. Una de las representaciones más expresivas y legibles son los conjuntos de reglas de la forma “si-entonces”. Cuando la evidencia se representa como un conjunto de pares (atributo-valor), las reglas tienen como premisas una conjunción de condiciones sobre los atributos de la forma $atributo_i = valor_{ij}$, donde $valor_{ij}$ es uno de los posibles

valores pertenecientes al dominio del atributo i , mientras que el consecuente es una etiqueta con la clase que se asignará a una instancia en caso de que se aplique la regla.

Veamos un ejemplo. Supongamos que un supermercado desea clasificar a sus clientes entre buenos y malos clientes. Para ello, dispone de una base de datos en la que almacena información acerca de los clientes y de la forma de pago de los mismos. La Tabla 17.1 muestra algunas de las tuplas de esta base de datos. Considerando únicamente esta evidencia algunas de las posibles reglas de clasificación podrían ser:

SI $N - \text{hijos} > 2$ ENTONCES $\text{Buen-cliente} = \text{si}$

SI $\text{Casado} = \text{no}$ Y $\text{Sexo} = h$ Y $N - \text{hijos} = 0$ ENTONCES $\text{Buen-cliente} = \text{si}$

Como podemos ver en el ejemplo, este tipo de reglas resultan tan comprensibles porque se construyen directamente usando las propiedades o atributos de los mismos datos que se han utilizado para inducir las reglas: si está casado, el número de hijos, etc.

Id	Casado	N-hijos	Sexo	Pago	Buen-cliente
1	si	3	m	Tarjeta	sí
2	no	0	h	Tarjeta	sí
3	no	1	m	Efectivo	no
4	si	4	m	Crédito	sí
5	si	2	h	Efectivo	no
6	no	1	m	Tarjeta	no
7	no	0	h	Efectivo	sí
8	no	0	h	Crédito	sí
9	no	1	h	Tarjeta	no
10	si	4	m	Crédito	sí

Tabla 17.1: Algunas tuplas de una base de datos de clientes de un supermercado.

En este capítulo estudiamos algunos de los algoritmos más ampliamente utilizados para el aprendizaje de estos conjuntos de reglas. Básicamente, las diferentes aproximaciones se clasifican, atendiendo al principio empleado en la generación de las reglas, en dos grupos:

- Los algoritmos basados en el principio “divide y vencerás” (también denominados “basados en el criterio de partición” (en inglés, *splitting*)): consiste en ir partiendo sucesivamente los datos en función del valor de un atributo seleccionado cada vez. Ésta es la estrategia que siguen los algoritmos de inducción de árboles de decisión que estudiaremos en la siguiente sección.
- Los algoritmos basados en el principio de “separa y vencerás” (también denominados “basados en el criterio de cobertura” (en inglés, *covering*)): consiste en encontrar condiciones de las reglas que cubran la mayor cantidad de ejemplos de una clase y la menor del resto de las clases (generalmente, se pretende cubrir una sola clase). Estas técnicas se estudiarán en la sección 17.3.

Si representamos gráficamente el efecto de cada tipo de estrategia sobre el espacio de las evidencias, se puede observar que una estrategia basada en el criterio de partición parte el espacio en regiones cuyos límites son líneas paralelas a los ejes, cada

una de las cuales corresponde a una condición sobre un atributo. Sin embargo, una estrategia basada en cobertura produce agrupamientos de ejemplos, cada uno correspondiente a una regla. La Figura 17.1 muestra estas particiones para una evidencia descrita por dos atributos (un espacio bidimensional).

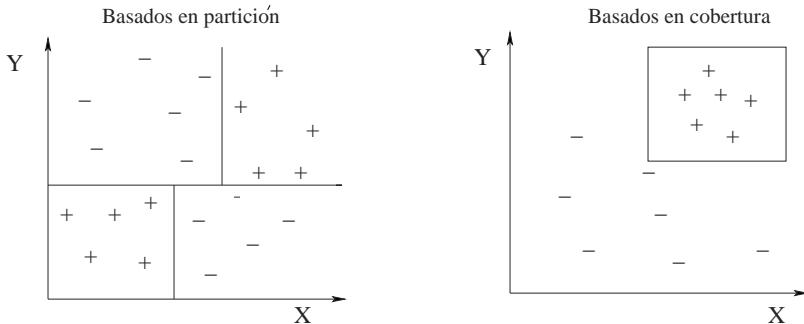


Figura 17.1: Clasificación basada en partición versus cobertura.

El resto del capítulo está organizado de la siguiente forma. En las dos secciones siguientes se presentan los métodos de aprendizaje de árboles de decisión y de reglas por cobertura, respectivamente. La sección 17.4 introduce algunas estrategias para reestructurar y optimizar conjuntos de reglas. La sección 17.5 muestra la inducción de árboles de regresión y agrupamiento, así como la utilización de árboles de decisión como estimadores de probabilidad. La sección 17.6 presenta otros algoritmos relacionados con los árboles de decisión. El capítulo concluye revisando algunos sistemas y con lecturas recomendadas y ejercicios.

17.2 Inducción de árboles de decisión

La inducción de árboles de decisión es uno de los métodos de aprendizaje más sencillos de entender y usar. Debido a esto, los árboles de decisión se han empleado para diferentes tareas de minería de datos, como la clasificación [Quinlan, 1986], la regresión [Breiman y otros, 1984] y el agrupamiento [Liu y otros, 2000].

Un árbol de decisión es una representación gráfica de un procedimiento para clasificar o evaluar un concepto. Por ejemplo, determinar a qué clase pertenecen las moléculas a partir de los átomos que las constituyen y sus enlaces, o determinar si se concede o no un crédito a un cliente a partir de ciertos datos como su sueldo, si ya tiene otros créditos, etc. Intuitivamente, un árbol de decisión es una colección de condiciones organizadas jerárquicamente. Formalmente, un árbol de decisión es un árbol donde cada nodo representa una condición o test sobre algún atributo y cada rama que parte de ese nodo corresponde a un posible valor para ese atributo. Finalmente, las hojas son las clases. Para clasificar una instancia se comienza en el nodo raíz, se

aplica el test al atributo especificado por este nodo y se sigue la rama que corresponde al valor que dicho atributo tiene en el ejemplo. Este proceso se repite hasta alcanzar una de las hojas la cual indica la clase de la instancia. Si el atributo clase es discreto el árbol de decisión recibe el nombre de *árbol de clasificación*, mientras que si la clase toma valores en un rango continuo entonces se dice que el árbol es de *regresión*.

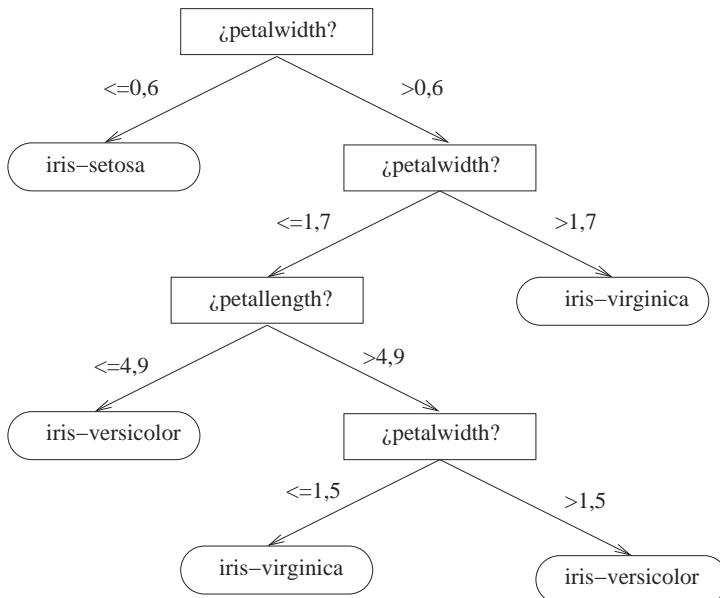


Figura 17.2: Árbol de decisión para clasificar plantas iris.

Una de las grandes ventajas de los árboles de decisión es que, en su forma más general, las opciones posibles a partir de una determinada condición son excluyentes. Esto permite analizar una situación y, siguiendo el árbol de decisión apropiadamente, llegar a una sola acción o decisión a tomar.

Consideremos, por ejemplo, la clasificación de plantas iris (conjunto de datos *Iris* del repositorio de datasets de aprendizaje automático UCI [Newman y otros, 1998]). Este problema consiste en determinar de qué tipo de planta iris se trata (*iris setosa*, *iris versicolor* o *iris virginica*) en función de las características del ancho y largo de su pétalo y/o sépalo. En la Figura 17.2, se muestra una representación en forma de árbol de decisión para este problema. Para clasificar una nueva instancia, por ejemplo (*sepallength* = 5,3, *sepalwidth* = 3,1, *petallength* = 1,4, *petalwidth* = 1,8), recorreríamos la rama de la derecha y concluiríamos que la planta se clasifica de tipo *iris virginica*.

En general, cada camino desde la raíz a una hoja corresponde a una conjunción de tests sobre los atributos, mientras que todo el árbol corresponde a una disyunción de

tales conjunciones. Esto es directamente expresable como un conjunto de reglas del tipo SI_ENTONCES_SI-NO donde las condiciones son la conjunción de tests sobre los atributos y las hojas son los consecuentes. Por ejemplo, el árbol de decisión de la figura anterior se puede representar como el conjunto de reglas de la Figura 17.3:

```

SI petalwidth <= 0,6 ENTONCES Iris - setosa
SI NO // petalwidth > 0,6
| SI petalwidth <= 1,7
| | SI petallength <= 4,9 ENTONCES Iris - versicolor
| | SI NO // petallength > 4,9
| | | SI petalwidth <= 1,5 ENTONCES Iris - virginica
| | | SI NO Iris - versicolor // petalwidth > 1,5
| | SI NO Iris - virginica // petalwidth > 1,7

```

Figura 17.3: Conjunto de reglas para el árbol de decisión de la Figura 17.2.

El esquema básico que siguen la mayoría de los algoritmos de inducción de árboles de decisión consiste en construir el árbol de forma top-down, desde la raíz a las hojas (*Top-Down Induction of Decision Trees, TDIDT* [Quinlan, 1986]) (véase el Algoritmo 17.1). Esta aproximación aplica una estrategia “divide-y-vencerás” en la que inicialmente se tienen todos los ejemplos (esto es, el conjunto de datos de entrenamiento), y a cada paso se selecciona el mejor atributo como test del nodo, se distribuyen los ejemplos entre los distintos nodos descendientes y se vuelve a proceder como en el nodo padre. De esta forma el árbol va creciendo. Este proceso continúa hasta que todos los ejemplos en un nodo son de la misma clase. En ese caso no es necesario seguir partiendo el nodo, el cual constituye una hoja del árbol y cuyo valor para la clase es el valor de ese atributo en los ejemplos que hay en él. Como en cada paso se van partiendo los datos; a esta forma de proceder también se conoce como estrategia “por partición”.

Algoritmo 17.1 Algoritmo TDIDT.

Procedimiento TDIDT (E : conjunto de ejemplos)

Crear raiz R ;

Crea-Arbol (R, E);

Nótese que este algoritmo parte de la hipótesis de que los datos son perfectos, es decir que no existen valores erróneos o ruido. Por ello, la condición que indica que hemos alcanzado una hoja es que todos los ejemplos sean de la misma clase, es decir, las hojas son puras. Esta condición puede resultar demasiado restrictiva en aplicaciones reales, donde puede ocurrir que sea difícil, cuando no imposible, eliminar completamente el ruido. También puede ser que no interese tener hojas puras ya que podría dar lugar a modelos sobreajustados a los datos de entrenamiento o excesivamente complejos. En la sección 17.4 se presentan algunas técnicas para relajar esta condición, permitiendo que las hojas del árbol sean razonablemente puras.

Las dos funciones más importantes en el algoritmo de inducción de árboles de decisión son la generación de posibles particiones (Generar-particiones(\cdot)) y la se-

Algoritmo 17.2 Procedimiento Crea-Árbol.

```
Procedimiento Crea-Árbol( $N$ : nodo ,  $E$ : conjunto de ejemplos)
    si todos los ejemplos  $E$  son de la misma clase  $c$  entonces
        Asignar la clase  $c$  al nodo  $N$ ;
        SALIR; {Esta rama es pura, ya no hay que seguir partiendo.  $N$  es hoja}
    si no
         $Particiones =$  Generar-particiones( $E$ );
         $MejorParticion :=$  Seleccionar-mejor-partición( $Particiones$ );
        para cada condición  $i$  de  $MejorParticion$  hacer
            Añadir un nodo hijo  $i$  a  $N$  y asignar los ejemplos consistentes ( $E_i$ ) a cada hijo;
            Crea-Árbol ( $i$ ,  $E_i$ ). {Realizar el mismo procedimiento global con cada hijo};
        fin para
    fin si
```

lección de la mejor partición (Seleccionar-mejor-particion(\cdot)). De hecho, los distintos algoritmos de partición, como por ejemplo ID3 [Quinlan, 1986], CART [Breiman y otros, 1984], ASSISTANT [Cestnik y otros, 1987], o C4.5 [Quinlan, 1993], se distinguen precisamente en estas funciones. Su importancia se debe a que una de las características del algoritmo es que una vez elegida la partición ya no se puede cambiar, aunque posteriormente comprobemos que ha sido una mala elección (el árbol inducido no es bueno); dicho de otra forma, la generación del árbol sigue una estrategia voraz. Por otra parte, cuantas más particiones permitamos más posibilidades habrán de crear árboles que reflejen mejor los datos, sin embargo, estos árboles también serán más complejos. Por ello, la mayoría de algoritmos de aprendizaje de árboles de decisión sólo permiten dos tipos de particiones:

- Particiones sobre atributos nominales: si un atributo x_i es nominal y sus valores son $\{v_1, v_2, \dots, v_k\}$ entonces las posibles particiones son de la forma $x_i = v_j$ siendo $1 \leq j \leq k$. Por ejemplo, el atributo *Casado* en el ejemplo de la Tabla 17.1 es nominal y sus posibles valores son *{sí,no}*. Por lo tanto, la partición para este atributo es (*Casado = no*, *Casado=sí*). Algunos algoritmos sólo generan árboles de decisión binarios, por lo que todas las particiones tienen que ser también binarias. Para ello, si el atributo tiene k posibles valores, se generan k particiones binarias de la forma $(x_i = v_j, x_i \neq v_j)$, $1 \leq j \leq k$.
- Particiones sobre atributos numéricos: generalmente, si el atributo x_i es numérico (continuo) su rango se parte en dos intervalos, tales que las particiones son de la forma $(x_i \leq v_c, x_i > v_c)$ siendo v_c algún valor intermedio entre los valores máximo v_{max} y mínimo v_{min} que tiene el atributo x_i en la evidencia E . v_c es llamado el punto de corte y existen diversas aproximaciones para determinarlo. La más sencilla consiste en ordenar los ejemplos de E en orden creciente de acuerdo al valor del atributo x_i (posiblemente eliminando valores repetidos). A continuación, se calculan los puntos medios entre dos ejemplos consecutivos en la secuencia ordenada. Cada uno de estos puntos es candidato a ser punto de corte. Por ejemplo, consideremos el atributo numérico *N – hijos* del ejemplo

de la Tabla 17.1 cuyos valores son $\{3, 0, 1, 4, 2, 1, 0, 0, 1, 4\}$. Ordenando estos valores sin tener en cuenta los repetidos, obtenemos la secuencia $\{0, 1, 2, 3, 4\}$, por lo que los puntos de corte son $\{0,5, 1,5, 2,5, 3,5\}$. Una forma de reducir el número de puntos de corte es calcular el valor intermedio entre dos valores consecutivos del atributo continuo para los que difiere la clasificación. Por ejemplo, supongamos un atributo numérico A en un problema de clasificación de dos clases (*sí, no*), con los siguientes valores

A	10	15	25	30	35
Clase	<i>sí</i>	<i>sí</i>	<i>no</i>	<i>no</i>	<i>sí</i>

entonces, los puntos de corte son 20 y 32,5 correspondientes al valor medio entre 15 y 20, y entre 30 y 35.

Una vez encontrados los puntos de corte, se pueden aplicar diversos criterios para establecer el o los puntos óptimos [Elomaa y Rousu, 1996; Fayyad y Irani, 1992] reduciendo así la talla del árbol (por ejemplo, el criterio de ganancia de información que explicaremos más adelante en esta misma sección, ya que se utiliza también como criterio para seleccionar la mejor partición). Por ejemplo, de los puntos de corte correspondientes al atributo $N - \text{hijos}$ podríamos decidir que el valor 1,5 es el punto de corte óptimo con lo que la partición quedaría ($N - \text{hijos} \leq 1,5, N - \text{hijos} > 1,5$). De igual modo, se podría también considerar el usar más de un punto de corte. Siguiendo con el ejemplo, si los puntos de corte óptimos son los valores 1,5 y 3,5, entonces la partición es ($N - \text{hijos} \leq 1,5, 1,5 < N - \text{hijos} \leq 3,5, N - \text{hijos} > 3,5$).

Volvamos otra vez al Algoritmo 17.1. Observamos que, una vez determinadas las posibles particiones, el siguiente paso es elegir la mejor partición para un nodo. Como hemos dicho, esto se hace de forma voraz, ordenando las particiones de acuerdo a una función heurística y eligiendo la que está mejor situada en el ranking. Se han sugerido muchas heurísticas, la mayoría de las cuales se basan en minimizar la *entropía*.

La *entropía* es una medida usada habitualmente para caracterizar el grado de (im)pureza de los ejemplos. Dada un conjunto de ejemplos E de un problema de clasificación con c clases, su entropía se define como

$$\text{Entropía}(E) = \sum_{i=1}^c -p_i \cdot \log_2 p_i \quad (17.1)$$

donde p_i es la probabilidad de que un ejemplo sea de la clase i (número de ejemplos de clase i dividido por el número de ejemplos de E).

Una forma cuantitativa de ver el efecto de partir los datos por un atributo en particular es usar el criterio llamado *Ganancia de Información* (*Information Gain*), el cual simplemente calcula la reducción esperada de la entropía por partir los ejemplos de acuerdo a ese atributo.

Formalmente, la ganancia de información, $Gain(E, x)$, de un conjunto de ejemplos E en relación al atributo x , se define como

$$Gain(E, x) = Entropía(E) - \sum_{v \in V(x)} p_v \cdot Entropía(E_v) \quad (17.2)$$

donde $V(x)$ es el conjunto de valores del atributo x , E_v es el subconjunto de E cuyo atributo x tiene el valor v y p_v es la probabilidad de que el atributo x tome el valor v ($|E_v|/|E|$). Maximizando la ganancia de información uno espera obtener árboles más pequeños al seleccionar en un nodo dado aquel atributo que permite distribuir los ejemplos en grupos (nodos) lo más homogéneos (puros) posible. En la sección 17.10 de ejercicios resueltos hay un ejemplo de generación de un árbol de decisión utilizando la ganancia como criterio de partición.

No obstante, la ganancia de información es una medida que no refleja si el atributo x distribuye los ejemplos de forma uniforme o no. Para incluir esta información se ha propuesto el criterio “ratio de ganancia” (*gain ratio* [Quinlan, 1986]) el cual es una modificación de la ganancia de información definida como

$$GainRatio(E, x) = \frac{Gain(E, x)}{SplitInformation(E, x)}$$

donde E es un conjunto de ejemplos, x es el atributo seleccionado y la entropía de E con respecto a x está representada por la función $SplitInformation(E, x)$.

Otro criterio es la heurística *Gini* [Breiman y otros, 1984] la cual es similar a la ganancia de información pero reemplazando la entropía de un conjunto de ejemplos E por

$$g(E) = 1 - \sum_{i=1}^c p_i^2$$

Por lo tanto, la calidad de las particiones inducidas por un atributo x queda establecida como

$$Gini(E, x) = g(E) - \sum_{v \in V(x)} p_v \cdot g(E_v) \quad (17.3)$$

Hay muchos otros criterios pero los incluidos en esta sección son los más populares en la inducción de árboles de decisión y han sido comparados no sólo teóricamente sino también experimentalmente (por ejemplo, en [Buntine y Niblett, 1992]).

17.3 Aprendizaje de reglas por cobertura

Los árboles de decisión no son la única forma de generar modelos en forma de reglas. En esta sección vamos a presentar una serie de algoritmos que construyen directamente las reglas siguiendo una estrategia de cobertura secuencial que consiste en generar las reglas de una en una, eliminando cada vez los ejemplos cubiertos por la nueva regla. Este proceso continúa hasta llegar al conjunto final de reglas. Por ello,

a todos estos métodos se les conoce como “algoritmos de cobertura secuencial” y están basados en la familia de algoritmos llamados *AQ* [Michalski y Larson, 1980; Michalski y otros, 1986].

Las principales diferencias entre estos algoritmos y los árboles de decisión son:

- En un árbol de decisión las reglas son excluyentes entre sí ya que provienen de condiciones diferentes (distintas particiones). Sin embargo, en los algoritmos por cobertura las condiciones de las reglas pueden solaparse, ya que cada una de ellas se genera de forma independiente y sin tener en cuenta las reglas anteriormente construidas.
- En un árbol de decisión cada nuevo ejemplo es clasificado en una única clase, ya que sólo se podrá aplicar una regla para su clasificación. En cambio, en los algoritmos por cobertura un mismo ejemplo puede ser clasificado como de dos o más clases diferentes cuando se puede aplicar más de una regla.

En general, los algoritmos basados en cobertura siguen el esquema mostrado en el Algoritmo 17.3. En principio partimos de un conjunto de ejemplos que podemos separar en positivos (los que el modelo debe cubrir) y negativos (los que no se tienen que cubrir). Entonces se van generando reglas de una en una hasta que no quedan ejemplos positivos que cubrir.

Algoritmo 17.3 Función Cobertura-secuencial.

Función Cobertura-secuencial (E_p, E_n : conjunto de ejemplos positivos y negativos)
Reglas= \emptyset ;
mientras $E_p \neq \emptyset$ Y *No-parada* **hacer**
 NRegla=Generar-una-regla(E_p, E_n);
 Reglas=*Reglas* \cup *NRegla*;
 $E_p = E_p -$ Ejemplos cubiertos por (*NRegla*);
fin mientras
devolver *Reglas*

Por lo tanto, la función más importante en este proceso es Generar-una-regla(.). Una forma efectiva de construir las reglas es organizar el espacio de hipótesis y hacer una búsqueda de lo general a lo específico. No es necesario recorrer todo el espacio, sino solamente aquellos caminos más prometedores. Más concretamente, comenzamos con la regla cuya condición es la vacía (ésta es la regla más general). Entonces vamos añadiendo condiciones de la forma (*atributo* = *valor*) de acuerdo a cierta medida de calidad que se define en términos de los ejemplos de entrenamiento (por ejemplo, podemos usar como medida la precisión, la cual puede calcularse como el número de ejemplos correctamente cubiertos por la regla dividido por el número de ejemplos a los que se puede aplicar la misma, o bien seleccionar la condición que permite cubrir la menor cantidad de ejemplos negativos, o cualquiera de los criterios vistos en la sección anterior como la ganancia de información). Este proceso continúa hasta tener una regla de calidad aceptable. Este esquema se refleja en el Algoritmo 17.4 en donde se ha utilizado como criterio de calidad la cobertura de ejemplos negativos. Nótese

que, en general, en ambos algoritmos se pueden utilizar criterios de parada adicionales (representados por la condición *No – parada*) para permitir un cierto margen de error en las reglas.

Algoritmo 17.4 Función Generar-una-regla.

Función Generar-una-regla (E_p, E_n : conjunto de ejemplos positivos y negativos)

Condición= \emptyset ;

$E_n\text{-act}=E_n$;

mientras $E_n - act \neq \emptyset$ Y *No – parada* **hacer**

Cond=generar una condición según criterio (eliminar el mayor número de ejemplos negativos);

Condición = *Condicion* $\cup \{Cond\}$;

$E_n\text{-act}=E_n\text{-act} -$ Ejemplos negativos no cubiertos por Condición;

fin mientras

devolver rule *Si Condición Entonces predicción {predicción es la clase de los ejemplos positivos cubiertos}*

Veamos un ejemplo. Consideremos la siguiente evidencia (véase Tabla 17.2) correspondiente al problema de recomendar lentes de contacto (dataset *Lenses* del repositorio UCI). Cada instancia viene descrita por cuatro atributos nominales (la edad discretizada en *joven*, *pre – presbicia* y *presbicia*; el diagnóstico, que puede ser *miope* o *hipermétrope*; la presencia o no de astigmatismo; si la producción de lágrimas es *reducida* o *normal*) más el atributo clase (que indica el tipo de recomendación: lentilla *dura*, lentilla *blanda* o ninguna), a los que hemos añadido un identificador para designar los ejemplos. Para aprender las reglas que definen la clase *dura*, tomaremos como ejemplos positivos las instancias {4, 8, 12, 20}, y el resto (que corresponden a las otras dos clases) como ejemplos negativos.

Una vez identificados los ejemplos negativos y los positivos, generamos todos los pares (*atributo* = *valor*) y calculamos cuántos ejemplos negativos cubre cada test (véase Tabla 17.3). El mejor par es (*Edad* = *joven*) que sólo cubre 6 ejemplos negativos. Descartamos los ejemplos no cubiertos que son {9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24}. Procediendo de igual forma con los ejemplos restantes, la siguiente condición seleccionada es (*Astigmatismo* = *si*) la cual permite eliminar los ejemplos negativos {1, 2, 5, 6}. Finalmente, la última condición es (*Lágrimas* = *normal*) que no cubre ningún ejemplo negativo de los que quedan, por lo que se eliminan todos. En consecuencia, la regla construida es

SI *Edad* = *joven* Y *Astigmatismo* = *si* Y *Lágrimas* = *normal* ENTONCES
Lentilla = *dura*

Esta regla cubre los ejemplos positivos 4 y 8, por lo que el algoritmo continuaría generando reglas para cubrir los ejemplos positivos restantes, en nuestro caso el 12 y el 20.

Otra característica del Algoritmo 17.4 es que la generación de las reglas sigue una estrategia voraz, en el sentido de que una vez seleccionada una condición (la que se

Instancia	Edad	Diagnóstico	Astigmatismo	Lágrimas	Lentilla
1	joven	miope	no	reducida	ninguna
2	joven	miope	no	normal	blanda
3	joven	miope	sí	reducida	ninguna
4	joven	miope	sí	normal	dura
5	joven	hipermétrope	no	reducida	ninguna
6	joven	hipermétrope	no	normal	blanda
7	joven	hipermétrope	sí	reducida	ninguna
8	joven	hipermétrope	sí	normal	dura
9	pre-presbicia	miope	no	reducida	ninguna
10	pre-presbicia	miope	no	normal	blanda
11	pre-presbicia	miope	sí	reducida	ninguna
12	pre-presbicia	miope	sí	normal	dura
13	pre-presbicia	hipermétrope	no	reducida	ninguna
14	pre-presbicia	hipermétrope	no	normal	blanda
15	pre-presbicia	hipermétrope	sí	reducida	ninguna
16	pre-presbicia	hipermétrope	sí	normal	ninguna
17	presbicia	miope	no	reducida	ninguna
18	presbicia	miope	no	normal	ninguna
19	presbicia	miope	sí	reducida	ninguna
20	presbicia	miope	sí	normal	dura
21	presbicia	hipermétrope	no	reducida	ninguna
22	presbicia	hipermétrope	no	normal	blanda
23	presbicia	hipermétrope	sí	reducida	ninguna
24	presbicia	hipermétrope	sí	normal	ninguna

Tabla 17.2: Evidencia del problema de recomendación de lentes de contacto.

considera que es mejor en ese momento) la regla sigue creciendo con los descendientes de la misma (esta condición nunca se elimina). Es decir, la regla se construye siguiendo elecciones subóptimas que no siempre conducen a una solución global óptima. Una forma de resolver este problema es la planteada por el algoritmo CN2 [Clark y Niblett, 1989]. Este algoritmo propone una búsqueda *beam* en la que en cada paso se seleccionan las k mejores condiciones y se sigue con los descendientes de todas ellas. Luego el conjunto resultante es nuevamente reducido a los k más prometedores.

Atributo	Valor	Ejemplos negativos cubiertos
edad	joven	1,2,3,5,6,7
edad	pre-presbicia	9,10,11,13,14,15,16
edad	presbicia	17,18,19,21,22,23,24
Diagnóstico	miope	1,2,3,9,10,11,17,18,19
Diagnóstico	hipermétrope	5,6,7,13,14,15,16,21,22,23,24
Astigmatismo	sí	3,7,11,15,16,19,23,24
Astigmatismo	no	1,2,5,6,9,10,13,14,17,18,21,22
Lágrimas	reducida	1,3,5,7,9,11,13,15,17,19,21,23
Lágrimas	normal	2,6,10,14,16,18,22,24

Tabla 17.3: Ejemplos negativos cubiertos por cada test *atributo–valor* para el ejemplo de la tabla 17.2.

Tal y como hemos mencionado al principio de esta sección, dado que las reglas se evalúan en cualquier orden y que las condiciones no son excluyentes entre sí puede

que exista más de una predicción para un ejemplo. Por ello, generalmente los sistemas generan las llamadas *listas de decisión*, las cuales se caracterizan porque las reglas se evalúan en el orden en que se han creado, impidiendo así el asignar más de un valor de la clase a un ejemplo.

Algunos otros sistemas de aprendizaje de reglas son 1R [Holte, 1993] el cual genera reglas con una única condición, y PRISM [Cendrowska, 1987] que genera reglas siguiendo el criterio de maximizar la relación entre ejemplos positivos cubiertos y ejemplos cubiertos en total a la hora de seleccionar las condiciones y que asume que no hay ruido en los datos.

17.4 Reestructuración de reglas

En las secciones anteriores hemos visto las técnicas más importantes de generación de árboles de decisión y conjuntos de reglas por cobertura. Sin embargo, estas técnicas, tal y como las hemos presentado, no son válidas para la obtención de modelos que sean útiles para su aplicación en la predicción de casos futuros, ya que generan modelos sobreajustados (*overfitting*) a los datos de entrenamiento, es decir modelos que se ajustan demasiado a la evidencia y, como consecuencia, se comportan mal para nuevos ejemplos ya que, en la mayoría de los casos, el modelo es solamente una aproximación del concepto objetivo del aprendizaje.

Consideremos el caso del aprendizaje de árboles de decisión. Las técnicas comentadas van recursivamente particionando los datos mediante la inclusión de ramas hasta encontrar un subconjunto de datos con la misma clase, entonces esa rama finalizará en ese momento con una hoja que indicará el valor de la clase correspondiente y en ella no se continuará la construcción. Este comportamiento significa que, dado un conjunto de datos de entrenamiento que representa una parte del problema a ser resuelto, los algoritmos de aprendizaje de árboles de decisión inducen un árbol que es capaz de clasificar correctamente todos los datos del conjunto de entrenamiento (salvo presencia de inconsistencias en los datos). Por lo tanto, en muchas ocasiones nos encontraremos con árboles de decisión de tamaño desorbitado con muy pocas instancias en la mayoría de las hojas. Este fenómeno se debe a un sobreajuste del árbol de decisión al conjunto de entrenamiento.

Por otra parte, si el conjunto de datos de entrenamiento contiene elementos con ruido (errores en los atributos o incluso en las clases) estos datos no son detectados y se consideran como correctos, por lo que la precisión en la predicción de casos venideros estará gravemente afectada.

Para resolver este tipo de problemas se suele aplicar al árbol de decisión un proceso donde se eliminan las partes que parecen demasiado específicas. A este proceso se le denomina poda. Este término es muy acertado, ya que las técnicas de poda lo que suelen hacer es eliminar aquellas partes del árbol que resultan poco útiles para él.

La aplicación de técnicas de poda reduce, generalmente, el tamaño del árbol de decisión, evitando el problema del sobreajuste. Sin embargo, una poda demasiado generosa puede provocar que el árbol resultante sea demasiado general o simple, cayendo en el problema contrario, el subajuste de datos. En la Figura 17.4 podemos observar

el efecto habitual en la precisión de un árbol de decisión dependiendo del nivel de aplicación de poda. Mientras que en el conjunto de entrenamiento la poda reduce el nivel de precisión (como es de esperar), en el conjunto de test la mejor precisión se obtiene frecuentemente con niveles medios de poda. Una poda excesiva provoca una simplificación excesiva del modelo, y consecuentemente, la degradación de su precisión. Para cada problema en concreto existe un grado de poda que es óptimo y que se puede estimar mediante el uso de datos de validación, si se dispone de ellos¹.

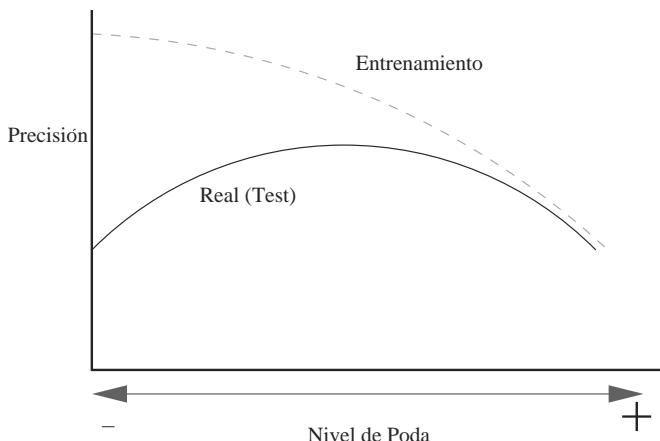


Figura 17.4: Efecto de la poda en la precisión.

Dependiendo del momento de aplicación de la fase de poda, podemos distinguir dos familias de métodos de poda:

- **Pre-poda:** Un método de poda se considera pre-poda si el proceso de selección de ramas a ser eliminadas se efectúa al mismo tiempo que el árbol se va generando. Es decir, la poda se incluye dentro del proceso de construcción del árbol de decisión.
- **Post-poda:** Un método de poda se considera post-poda si el proceso inspecciona y determina las ramas a ser eliminadas una vez que el árbol se ha completado. De esta manera, la fase de poda se realiza posteriormente a la construcción del árbol.

Los métodos de pre-poda suelen ser más eficientes, sin embargo, los métodos de post-poda suelen obtener árboles de decisión más precisos, ya que tienen la ventaja de conocer completamente el árbol de decisión a la hora de ser aplicados.

¹En aprendizaje automático se denomina conjunto de validación a un conjunto de datos utilizado para refinar el modelo aprendido desde el conjunto de entrenamiento (como por ejemplo, para fijar el nivel de poda), mientras que el conjunto de test es el conjunto de datos utilizado para evaluar la calidad del modelo.

Brevemente, algunas de las técnicas de poda más conocidas y utilizadas son las siguientes [Esposito y otros, 1997]:

- **Poda de error reducido:** Método de poda propuesto por Quinlan [Quinlan, 1987]. Esta técnica busca el grado óptimo de poda del árbol utilizando un conjunto de datos de validación. Esta técnica tiende a sobreponer los árboles además de la desventaja de tener que utilizar parte de los datos disponibles para el conjunto de validación de la poda.
- **Poda de error pesimista:** Método de poda también propuesto por Quinlan [Quinlan, 1987]. Esta técnica convierte un subárbol en hoja, si una estimación pesimista del error (basada en la distribución binomial) considerando el nodo como hoja es menor del valor estimado si lo tomamos como subárbol.
- **Poda basada en coste-complejidad:** Método de poda también conocido por método de poda CART [Breiman y otros, 1984]. En este caso, el error estimado considera la suma ponderada de su complejidad y su error sobre los ejemplos de entrenamiento, mientras que los ejemplos de validación son usados principalmente para determinar una ponderación adecuada.
- **Poda basada en error:** Técnica empleada en el sistema C4.5 [Quinlan, 1993]. Se considera una versión del método de poda de error pesimista, ya que en este caso se utiliza una estimación del error todavía más pesimista. Además, tiene la ventaja con respecto a otros métodos, que permite reemplazar un subárbol por una de sus ramas. De esta manera, pueden eliminarse particiones intermedias que no son demasiado útiles.

En [Esposito y otros, 1997] se comparan éstas y otras técnicas populares de poda de árboles de decisión. De los experimentos de este trabajo, los autores concluyen que, en prácticamente todos los problemas estudiados, la utilización de la poda sirve para mejorar la calidad (medida en precisión) de los árboles aprendidos.

Existen otras aproximaciones a la poda de árboles de decisión. Por ejemplo, En [Mehta y otros, 1995] se utiliza el principio MDL/MML (*Minimum Description Length/Minimum Message Length principle*) [Barron y otros, 1998; Wallace y Dowd, 1999] para estimar la complejidad del árbol con respecto a la evidencia, y de esta manera seleccionar el modelo que explica de manera más sencilla los datos de entrenamiento (principio de Occam).

Aparte de la poda, existen otros mecanismos de transformación de árboles de decisión que permiten mejorar su comportamiento generalmente denominados operadores de reestructuración. Por ejemplo, el algoritmo C4.5 realiza lo que se conoce como colapsamiento (*collapsing*) [Quinlan, 1993]. Otros operadores de modificación de la topología del árbol son la transposición, la transposición recursiva y la poda virtual [Utgoff y otros, 1997].

17.5 Otras aplicaciones de los árboles de decisión

Como se ha mencionado en la sección 17.2, aunque la aplicación más conocida de los árboles de decisión es la clasificación, existen otras variantes de los árboles de decisión que permiten que se puedan emplear para otras tareas como la regresión o el agrupamiento.

Por ejemplo, uno de los primeros sistemas de aprendizaje de árboles de decisión definidos, CART [Breiman y otros, 1984], permite el aprendizaje de modelos de clasificación (el valor a predecir es nominal), así como de regresión (el valor a predecir es numérico). En realidad, un árbol de regresión se construye de manera similar a un árbol de clasificación, salvo que los nodos hoja del árbol se etiquetan con valores numéricos. De esta manera, la calidad del árbol se mide en términos de una cierta medida de calidad a maximizar, por ejemplo la varianza de los ejemplos que caen en ese nodo respecto al valor asignado.

Una implementación sencilla de esta idea es el propio algoritmo CART. El método de aprendizaje realiza particiones binarias sobre los atributos de igual manera que lo visto para los árboles de clasificación, pero en este caso asignando una media y una varianza a cada nodo e intentando seleccionar las particiones que reduzcan las varianzas de los nodos hijos.

Una variante muy popular de los árboles de regresión consiste en considerar una función lineal en los nodos, en vez de una media y una desviación típica. En este caso, en primer lugar, para evaluar las particiones se puede utilizar, por ejemplo, el error cuadrático medio de la regresión lineal de los ejemplos que hayan caído en cada nodo. En segundo lugar, para los nodos hoja, la predicción se realiza utilizando el modelo lineal. Nótese que un modelo lineal se puede obtener de una forma relativamente sencilla y eficiente para un conjunto de ejemplos. También hay que destacar que esta modificación es directa si todos los atributos son numéricos. En caso de que existan también atributos nominales, se debería utilizar algún tipo de regresión lineal que trate con estos atributos nominales.

Por otra parte, también se han desarrollado algoritmos para el aprendizaje de árboles de decisión para su aplicación en problemas de agrupamiento. Una primera aproximación de estos algoritmos es la generación de particiones de manera que se separe entre zonas densas (zonas pobladas de ejemplos) y zonas poco densas (zonas donde se encuentran pocos ejemplos). Esto se sigue haciendo hasta que se llega a zonas muy densas o zonas muy poco densas, constituyendo entonces los nodos del árbol. Un refinamiento de esta idea es el método presentado por [Liu y otros, 2000], en el que se consideran todos los ejemplos de una clase E (por existentes) mientras que se añaden ejemplos ficticios N (por no existentes) uniformemente distribuidos en el espacio. El siguiente paso es simple, se utiliza un método de aprendizaje de árboles de decisión para clasificación (preferiblemente con poda). El resultado es que las reglas obtenidas para la clase E serán los clusters o grupos formados.

Evidentemente esta idea se puede refinar más y no es necesario crear realmente los puntos N , sino que se puede rediseñar el algoritmo, en particular, el criterio de partición para que los considere como si estuvieran allí. No obstante, si no disponemos de un algoritmo de agrupamiento mediante árboles de decisión, podríamos generar

los ejemplos ficticios nosotros mismos y utilizar algoritmos de clasificación clásicos, tales como CART o C4.5 para poder hacer el agrupamiento.

Por último, los árboles de decisión en su versión para clasificación pueden directamente ser utilizados como estimadores de probabilidades (también denominada clasificación suave). En este caso, los ejemplos tienen una etiqueta discreta, denominada clase. La diferencia es que el objetivo de los estimadores de probabilidades es más ambicioso: no se trata de determinar para cada nuevo ejemplo de qué clase es, sino determinar para cada nuevo ejemplo cuál es la probabilidad de que pertenezca a cada una de las clases. Un estimador de probabilidades es especialmente útil cuando se quiere acompañar las predicciones con una cierta fiabilidad, o se quiere combinar predicciones de varios clasificadores, o bien hacer un ranking de predicciones (por ejemplo, ordenar los clientes según su rendimiento).

La modificación de un árbol de clasificación para que sea un estimador de probabilidades es bien sencilla. Supongamos que tenemos tres clases: a , b y c . Para cada nodo hoja con una cardinalidad n (número total de ejemplos de entrenamiento que caen en ese nodo) tendremos un determinado número de ejemplos de cada clase: n_a , n_b y n_c . Si dividimos cada uno de estos valores por la cardinalidad total, tendremos una estimación de las probabilidades de las clases en ese nodo, es decir: $p_a = n_a/n$, $p_b = n_b/n$, y $p_c = n_c/n$. Este tipo de árboles de decisión modificados de esta manera se denominan *PETs* (*Probability Estimation Trees*).

Aunque la conversión es sencilla, las estimaciones de probabilidad obtenidas por los PETs de esta manera no son muy buenas comparadas con otros métodos. No obstante,

- El suavizado de frecuencias de las estimaciones de probabilidad de las hojas, como por ejemplo la corrección de Laplace o el m-estimado, mejoran significativamente las estimaciones. Con la corrección de Laplace las probabilidades se estiman de la siguiente forma:

$$p_i = \frac{(n_i + 1)}{n + C}$$

donde C representa el número de clases. La corrección m-estimado es muy similar, pero en este caso se introduce un parámetro m . El factor p representa la probabilidad a priori, la cual si se usa una distribución uniforme se define como $p = 1/C$. Nótese que en ese caso, si tomamos $m = C$ ambas correcciones coinciden:

$$p_i = \frac{(n_i + m \cdot p)}{n + m}$$

- La poda (o técnicas relacionadas de transformación) es generalmente perjudicial para la estimación de probabilidades. Los árboles sin podar dan los mejores resultados.

Teniendo en cuenta estas consideraciones, los árboles de decisión constituyen también una buena herramienta para la estimación de probabilidades [Ferri y otros, 2003; Provost y Domingos, 2003].

17.6 Extensiones de árboles y reglas de decisión

La fuerte popularidad de los sistemas de aprendizaje de reglas y árboles de decisión ha provocado que se hayan intentado combinar estas técnicas con otras técnicas de aprendizaje, dando lugar a métodos híbridos.

Se han definido hibridaciones tanto de los árboles de decisión para clasificación como para regresión. Por ejemplo, en clasificación, los árboles de decisión perceptrón [Utgoff, 1989] (véase capítulo 15) o los árboles de decisión multivariantes basados en máquinas lineales [Brodley y Utgoff, 1995] que en vez de utilizar particiones que involucran un único atributo permiten particiones que se basan en perceptrones (discriminantes lineales) o en conjuntos de perceptrones (denominados máquinas lineales). Esto permite extender la representación para tratar conceptos que no sean cuadriculares (los cuales dividen el espacio en regiones limitadas por líneas paralelas a los ejes, como en la imagen de la izquierda de la Figura 17.1) y así capturar mejor otro tipo de patrones basados en cualquier combinación de fronteras lineales. Todos estos métodos son una evolución de una variación del algoritmo CART para contemplar particiones lineales, denominado frecuentemente CART-LC. Otra extensión importante de los árboles de decisión es lo que se conoce como grafos de decisión [Dowe y otros, 1992]. Estos grafos son grafos acíclicos de decisión que permiten que distintas ramas de un árbol de decisión converjan. Los sistemas de aprendizaje de grafos permiten, en general, obtener modelos más compactos y compartir ejemplos por distintas vías, lo que puede tener efectos beneficiosos para evitar el sobreajuste y para ahorrar memoria. Los criterios de partición y de construcción deben redefinirse (por ejemplo utilizando el principio MML).

Un método similar es *Cascading* [Gama y Brazdil, 2000]. Este método permite mejorar las características de los árboles de decisión mediante la incorporación de nuevas particiones basadas en otras técnicas de aprendizaje. Concretamente, la estrategia de cascading se basa en utilizar otros métodos de aprendizaje (regresión lineal, análisis discriminante) sobre algunos de los atributos para crear nuevos atributos artificiales. Estos nuevos atributos son también utilizados por el árbol de decisión para crear los modelos. Los atributos artificiales representan conceptos intermedios, y permiten que el algoritmo de aprendizaje extienda la representación cuadriculada mediante la inclusión de divisiones oblicuas en el espacio de representación.

17.6.1 Métodos multiclassificadores

Otra familia de técnicas que permiten mejorar la precisión en la predicción de nuevos casos de los modelos aprendidos por algoritmos de aprendizaje de árboles y reglas de decisión son los métodos basados en la combinación de modelos [Dietterich, 2000a]. Estos métodos construyen un conjunto de hipótesis (*ensemble*), y entonces combinan las predicciones del conjunto de alguna manera (normalmente por votación) para clasificar ejemplos o para hacer regresión sobre ejemplos. La precisión obtenida por esta combinación de hipótesis supera, generalmente, la precisión de cada componente individual del conjunto. La combinación de modelos se ha desarrollado principalmente para modelos predictivos (clasificación y regresión), aunque ciertas ideas podrían

extenderse a modelos descriptivos. Estas técnicas no sólo se pueden aplicar sobre árboles o reglas de decisión; en realidad, se obtienen generalmente buenos resultados sobre cualquier tipo de modelo, independientemente de la técnica que se haya utilizado para su aprendizaje. Realmente, los factores que determinan la calidad del modelo combinado son en alto grado la precisión y la diversidad de los componentes del conjunto. Considérese el siguiente ejemplo [Dietterich, 2000a]. Dado un conjunto formado por tres clasificadores $\{h_1, h_2, h_3\}$, sea x un nuevo dato a ser clasificado. Si los tres clasificadores son similares, entonces cuando $h_1(x)$ sea erróneo, probablemente $h_2(x)$ y $h_3(x)$ también lo serán. Sin embargo, si los clasificadores son lo bastante diversos, los errores que cometan estarán poco correlacionados, y por tanto, cuando $h_1(x)$ sea erróneo, $h_2(x)$ y $h_3(x)$ podrían ser correctos, y entonces, si la combinación se realizase por votación mayoritaria, el conjunto combinado clasificaría correctamente el dato x .

Por lo tanto, la clave es obtener modelos que, siendo todos precisos, difieran lo suficiente entre sí. Una manera de obtener estos modelos es aprender cada uno de ellos con un subconjunto diferente de datos. Esta es la estrategia que utiliza *Bagging* [Breiman, 1996; Quinlan, 1996]. Este término deriva del mecanismo denominado *bootstrap aggregation*, mecanismo que genera subconjuntos de entrenamiento seleccionando aleatoriamente y con reemplazamiento (puede haber ejemplos repetidos) una muestra de m ejemplos de entrenamiento del conjunto original de entrenamiento formado por m ejemplos. Los subconjuntos nuevos son llamados *bootstrap replicates*. La Figura 17.5 describe el algoritmo de *Bagging* para clasificación. Desde cada subconjunto se aprende un modelo utilizando habitualmente para ello el mismo método de aprendizaje. Se podría esperar que si se utiliza siempre el mismo método, los modelos aprendidos desde cada subconjunto fuesen siempre similares, ya que existe un gran porcentaje de ejemplos comunes en todos los subconjuntos de entrenamiento. Sin embargo esto es generalmente falso. En especial, los árboles de decisión son muy sensibles al conjunto de entrenamiento, es decir, pequeños cambios en el conjunto de entrenamiento provocan que el algoritmo de aprendizaje de árboles de decisión construya un árbol bastante diferente.

Para la predicción de nuevos ejemplos, dado que hay un conjunto de clasificadores, en *Bagging* se efectúa una votación mayoritaria. Esto es, se elige la clase con mayor número de votos entre todos los clasificadores.

Otro método de combinación de modelos muy conocido es *Boosting* [Freund y Schapire, 1996; Quinlan, 1996]. En este caso, el mecanismo que emplea *Boosting* para aprender los diferentes modelos se basa en la asignación de un peso a cada ejemplo del conjunto de entrenamiento. En cada iteración, se aprende un modelo que minimiza la suma de los pesos de los ejemplos clasificados erróneamente. En la primera iteración todos los ejemplos del conjunto de entrenamiento reciben el mismo peso. Pero en las siguientes iteraciones, los errores previamente cometidos se utilizan para actualizar los pesos de los ejemplos del conjunto de entrenamiento, de manera que se incremente el peso de los ejemplos errados y se reduzca el peso de los ejemplos acertados. De esta forma, se consigue que el modelo que se aprenda en la siguiente iteración otorgue más relevancia a los ejemplos que los modelos anteriores habían clasificado erróneamente.

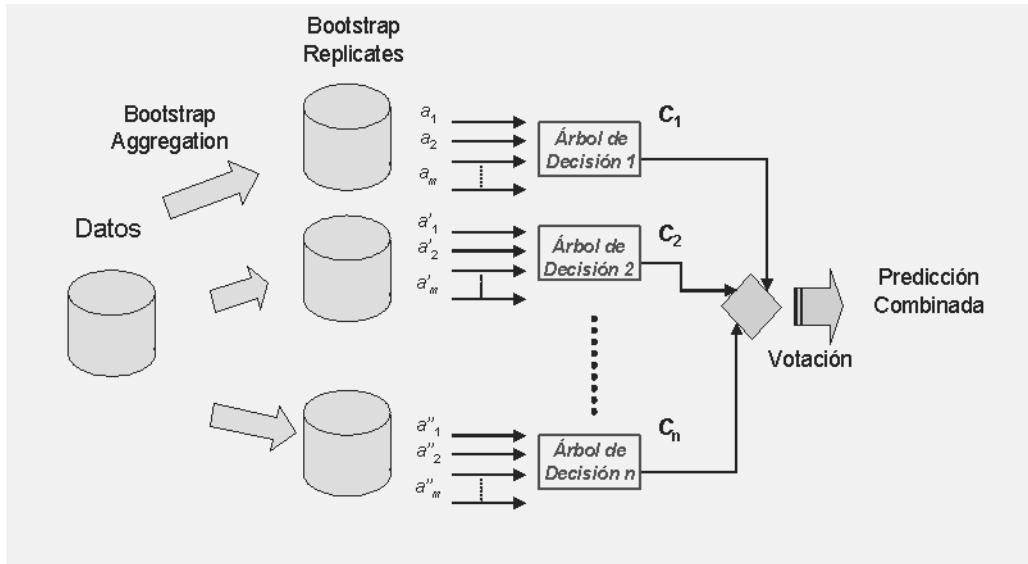


Figura 17.5: Bagging.

Boosting utiliza una estrategia bastante astuta, ya que construye los nuevos modelos tratando de corregir los errores cometidos previamente. Además, al contrario que el *Bagging* para el que la predicción de nuevos casos se establece por votación igualitaria, *Boosting* da más relevancia a los modelos que tienen un mejor comportamiento (menor error estimado), es decir, establece una votación ponderada.

Existen muchas variantes del algoritmo básico de *Boosting*, siendo probablemente *AdaBoost* [Freund y Schapire, 1996] la versión más popular. *AdaBoost* ha sido definido para problemas de clasificación aunque se puede adaptar a regresión modificando ligeramente su estrategia. La Figura 17.6 resume el algoritmo de *AdaBoost* para clasificación.

Existen varios trabajos que comparan experimentalmente estos y otros métodos multiclassificadores [Bauer y Kohavi, 1999; Dietterich, 2000b]. La conclusión de la mayoría de los trabajos es que *Boosting* obtiene mejor precisión en general, pero en los casos de problemas con ruido, *Bagging* es más robusto.

17.7 Sistemas y aplicabilidad

Debido al uso extendido del aprendizaje de reglas y árboles de decisión han proliferado gran cantidad de sistemas de aprendizaje de este tipo de modelos. Dejando aparte los sistemas pioneros y clásicos como CART [Breiman y otros, 1984], ID3 [Quinlan, 1983], C4.5 [Quinlan, 1993], o CN2 [Clark y Niblett, 1989] a los que ya

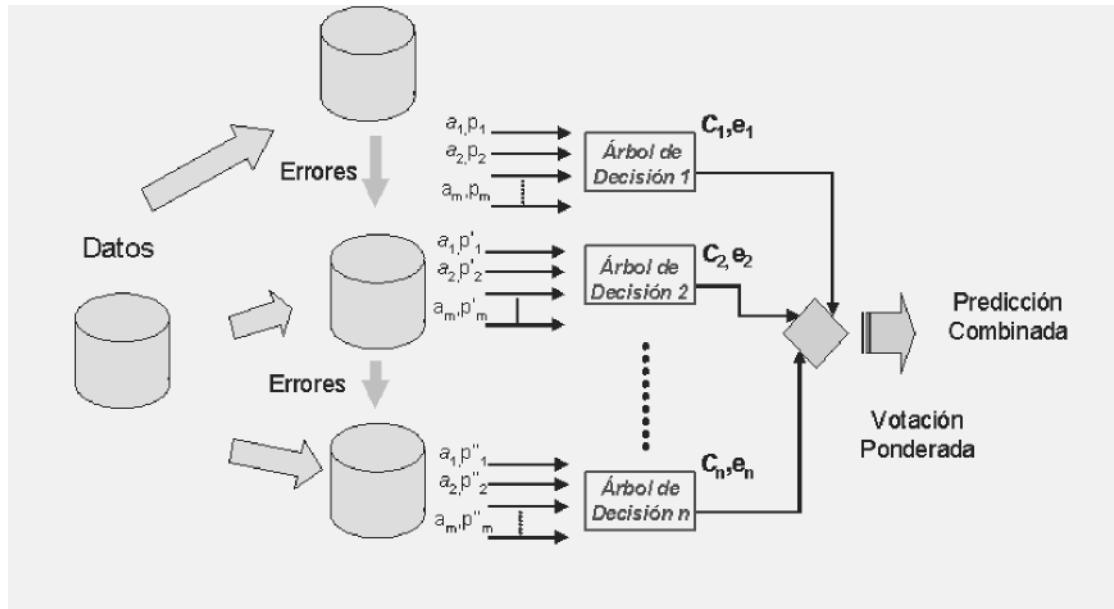


Figura 17.6: Boosting.

nos hemos referido a lo largo de este capítulo y que son generalmente sistemas de inducción bastante limitados por ser herramientas focalizadas, en esta sección vamos a presentar algunos sistemas de inducción de reglas y árboles de decisión integrados en paquetes de minería de datos o aplicaciones estadísticas. Estos sistemas tienen varias ventajas frente los sistemas clásicos. Por ejemplo, permiten frecuentemente una interacción sencilla con almacenes o bases de datos, o bien proporcionan mecanismos de evaluación de modelos, etc.

El primer software que vamos presentar es Weka [Witten y Frank, 2005]. Weka es un conjunto de librerías JAVA para la extracción de conocimiento desde bases de datos. Este software ha sido desarrollado en la Universidad de Waikato (Nueva Zelanda) bajo licencia GPL lo cual ha impulsado que sea una de las suites más utilizadas en el área en los últimos años.

Weka contiene una gran cantidad de métodos de varias tareas de minería de datos (especialmente clasificación y regresión). Pero además incluye un entorno de experimentación para comparar el rendimiento de los algoritmos utilizando técnicas estadísticas, así como un conjunto de filtros que permiten manipular los datos de origen: eliminar atributos, seleccionar atributos más relevantes, muestrear datos, etc.

Centrándose en la parte de los algoritmos de inducción de árboles y reglas de decisión, en la versión 3.4.8 podemos encontrar las técnicas disponibles en el entorno *Explorer* en el apartado *Classify* dentro de los subgrupos *Trees* y *Rules*. Entre los métodos disponibles destacamos J48 (una versión del algoritmo C4.5 [Quinlan, 1993]), así como el algoritmo ID3 [Quinlan, 1983].

En la Figura 17.7 podemos ver el árbol de decisión construido por la técnica J48 para el dataset “*house-vote*”. Este dataset pertenece al repositorio UCI [Newman y otros, 1998] y está compuesto por 435 instancias, cada una con 16 atributos. Cada instancia representa a un congresista de Estados Unidos y cada atributo corresponde a la respuesta que dio el congresista a una pregunta concreta. La clase indica si el congresista es republicano o demócrata. Nótese que en las hojas Weka informa sobre la distribución de la clase de los ejemplos que caen en esa misma hoja². Otros métodos de aprendizaje de árboles de decisión integrados en Weka son:

- **Decision Stump:** Árboles de decisión muy simples formados por tan sólo una rama, es decir un nivel. Varios trabajos han demostrado que modelos tan simples suelen tener un comportamiento aceptable [Holte, 1993].
- **LMT:** “Logistic Model Trees”, árboles de clasificación que incluyen funciones de regresión en las hojas [Landwehr y otros, 2005].

Además de estas técnicas de aprendizaje de árboles de decisión para clasificación, podemos encontrar también métodos para regresión. Por ejemplo:

- **Reptree:** Método basado en C4.5 que también admite problemas de regresión.
- **Decision Stump:** Árboles de decisión muy simples formados por tan sólo una rama, es decir un nivel. En la versión para problemas de regresión, las hojas se etiquetan con la media de los valores de los ejemplos que caen en ella.
- **M5P:** Técnica definida por Quinlan [Quinlan, 1992] que unifica el aprendizaje de árboles de decisión junto con la regresión.

Finalmente, en cuanto a árboles de decisión en Weka, destacamos el paquete “User Classifier”. Este paquete permite la construcción de un árbol de decisión mediante la inclusión manual de particiones por parte del usuario.

En cuanto a sistemas de aprendizaje de reglas, la suite está bastante más limitada. Entre estas técnicas encontramos:

- **PART:** En cada iteración construye un árbol parcial C4.5, y toma como regla la mejor hoja del árbol [Frank y Witten, 1998].
- **PRISM:** Genera reglas siguiendo el criterio de maximizar la relación entre ejemplos positivos cubiertos y ejemplos cubiertos en total a la hora de seleccionar las condiciones [Cendrowska, 1987].
- **ZeroR:** Implementa el método 0R. Esta técnica es muy simple, ya que construye un modelo que retorna la moda (clase mayoritaria) para problemas de clasificación, y la media para problemas de regresión.

Por último, podemos encontrar gran cantidad de técnicas de combinación de modelos en el paquete. Para ello debemos acudir al apartado meta. Resumimos algunos métodos.

²Las cifras no son exactas por el método que emplea J48 para tratar con atributos faltantes.

- **Bagging:** Versión del clásico algoritmo de Leo Breiman.
- **Boosting:** Dispone de varias versiones del algoritmo: paquetes “AdaBost”, “LogitBoost1” y “MultiBoostAB”.
- **Vote:** Implementación de la combinación mediante votación simple entre varios modelos.
- **Decorate:** Método de combinación de modelos, que consigue la diversidad entre los modelos base mediante la inclusión de ruido en los ejemplos [Melville y Mooney, 2003].
- **Stacking:** Versión del algoritmo de metaaprendizaje “Stacking”.

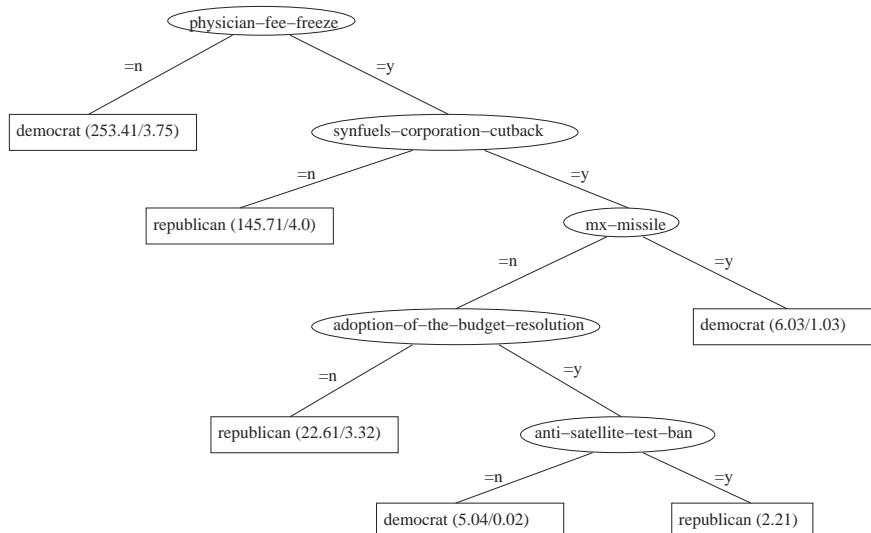


Figura 17.7: Árbol de decisión para el problema “House-vote”.

17.8 Lecturas recomendadas

A lo largo de este capítulo hemos referenciado los trabajos más importantes en el campo del aprendizaje de árboles y reglas de decisión. En esta sección recopilamos otras referencias que pueden ser de gran utilidad si el lector desea profundizar en la materia. Por desgracia, en castellano no existe demasiada literatura específica en árboles y reglas de decisión. Quizás la excepción sea el capítulo 11 del libro [Hernández y otros, 2004].

Sin embargo, en inglés podemos encontrar gran variedad de literatura. Destacamos el libro donde Quinlan expone con todo detalle su sistema C4.5 [Quinlan, 1993]. Existen capítulos dedicados a la inducción de árboles y reglas de decisión en libros genéricos de aprendizaje automático o minería de datos. Recomendamos: [Witten y Frank, 2005], [Mitchell, 1997] y [Han y Kamber, 2000].

Finalmente, existen varios artículos publicados a modo de “survey”, es decir, compilaciones o resúmenes de las contribuciones más importantes acaecidas en este área: [Breslow y Aha, 1997], [Murthy, 1998].

En cuanto a los métodos multiclassificadores, es muy recomendable la lectura de [Kuncheva, 2004], y en castellano aconsejamos el capítulo 18 del libro [Hernández y otros, 2004].

17.9 Resumen

En este capítulo hemos presentado las técnicas de aprendizaje de árboles y reglas de decisión. Estas técnicas pueden considerarse entre las más utilizadas dentro del área del aprendizaje automático para la obtención de modelos para problemas de clasificación y regresión. El motivo fundamental de esta popularidad es la sencillez y comprensibilidad de los modelos generados. Modelos que, por otra parte, suelen presentar una precisión comparable con los obtenidos con otras técnicas más complejas como pueden ser las redes neuronales o el aprendizaje bayesiano.

En resumen, las ventajas principales de las técnicas vistas en este capítulo son las siguientes:

- Son fáciles de usar por su directa comprensibilidad.
- Admiten atributos numéricos (continuos) y nominales (discretos).
- Son generalmente eficientes y existen versiones que permiten su escalabilidad a grandes volúmenes de datos.
- Son aplicables a varias tareas de minería de datos: clasificación, regresión, agrupamiento y estimación de probabilidades.
- Toleran el ruido, atributos no significativos y valores faltantes.
- Existen multitud de versiones y sistemas, muchos de ellos gratuitos.

En cuanto a las desventajas destacamos que, en general, no son tan precisos como otros métodos más complejos (redes neuronales o las máquinas de vectores de soporte). Además, son débiles (se conocen como *weak learners*) en el sentido que, debido a su carácter voraz, son dependientes de la muestra de ejemplos. Es decir, dos muestras distintas sobre la misma distribución pueden dar lugar a dos árboles muy diversos. Sin embargo, es precisamente esta debilidad o variabilidad la que permite que los árboles de decisión sean ideales para ser utilizados con técnicas de combinación de modelos, tal y como hemos comentado en la sección 17.6.1.

17.10 Ejercicios resueltos

17.1. Consideremos un problema de clasificación binario con clases 1 y 2, una evidencia formada por cinco ejemplos $E = \{e_1, e_2, e_3, e_4, e_5\}$ y A un atributo nominal, tales que

Identificador	e_1	e_2	e_3	e_4	e_5
A	a_1	a_3	a_2	a_2	a_2
Clase	2	2	1	2	2

Calcula la ganancia de información de A con respecto a E .

Solución: Para ello, primero obtenemos la probabilidad de cada clase: $p_1 = 1/5$ y $p_2 = 4/5$. El siguiente paso es calcular la entropía

$$\text{Entropía}(E) = -(1/5)\log_2(1/5) - (4/5)\log_2(4/5) = 0,46 + 0,26 = 0,72$$

Dividimos la evidencia de acuerdo a los valores del atributo A , $E_{a_1} = \{e_1\}$, $E_{a_2} = \{e_3, e_4, e_5\}$ y $E_{a_3} = \{e_2\}$, y calculamos los siguientes términos:

$$\begin{aligned} (|E_{a_1}|/|E|) \cdot \text{Entropía}(E_{a_1}) &= (1/5) \cdot [-(1/1)\log_2(1/1) - (0/1)\log_2(0/1)] = 0 \\ (|E_{a_2}|/|E|) \cdot \text{Entropía}(E_{a_2}) &= (3/5) \cdot [-(1/3)\log_2(1/3) - (2/3)\log_2(2/3)] = 0,552 \\ (|E_{a_3}|/|E|) \cdot \text{Entropía}(E_{a_3}) &= (1/5) \cdot [-(0/1)\log_2(0/1) - (1/1)\log_2(1/1)] = 0 \end{aligned}$$

Finalmente,

$$Gain(E, A) = 0,72 - (0 + 0,552 + 0) = 0,168$$

■

17.2. Construye el árbol de decisión correspondiente a la evidencia mostrada en la Tabla 17.2 usando la ganancia de información como criterio de partición.

Solución: De acuerdo a los Algoritmos 17.1 y 17.2, en cada nodo del árbol debemos calcular la ganancia de información por atributo usando las instancias en ese nodo y seleccionar el mejor de ellos, es decir, el atributo con el que se obtiene una mayor ganancia.

Nodo raíz: Para el nodo raíz consideramos las 24 instancias de la Tabla 17.2 y los atributos *edad*, *diagnóstico*, *astigmatismo* y *lágrimas*. Observa que no usamos los atributos *lentilla* e *instancia* ya que el primero es la clase (objetivo del árbol de decisión) y el segundo es, en realidad, un atributo que no pertenece al dominio del problema y que usamos únicamente para poder referirnos a los ejemplos más fácilmente. Como hay 15 ejemplos de clase *ninguna*, 5 de clase *blanda* y 4 de clase *dura*, la probabilidad de cada clase es

$$p_{\text{ninguna}} = 15/24 = 0,625 \quad p_{\text{blanda}} = 5/24 = 0,208 \quad p_{\text{dura}} = 4/24 = 0,167$$

y la entropía de la evidencia inicial E es

$$\begin{aligned} \text{Entropía}(E) &= -(15/24) \cdot \log_2(15/24) - (5/24) \cdot \log_2(5/24) - (4/24) \cdot \log_2(4/24) \\ &= 0,424 + 0,471 + 0,431 = 1,326 \end{aligned}$$

Ahora para cada atributo dividimos la evidencia de acuerdo a sus valores y calculamos el segundo término de la Ecuación 17.2:

- **Edad:** La evidencia queda partida en estos tres conjuntos:

$$\begin{aligned}E_{joven} &= \{1, 2, 3, 4, 5, 6, 7, 8\} \\E_{pre-presbicia} &= \{9, 10, 11, 12, 13, 14, 15, 16\} \\E_{presbicia} &= \{17, 18, 19, 20, 21, 22, 23, 24\}\end{aligned}$$

En E_{joven} hay 4 instancias de la clase *ninguna*, 2 instancias de la clase *blanda* y 2 de la clase *dura*. Análogamente, en $E_{pre-presbicia}$ hay 5 instancias de la clase *ninguna*, 2 instancias de la clase *blanda* y 1 de la clase *dura*; y, finalmente, en $E_{presbicia}$ hay 6 instancias de la clase *ninguna*, 1 instancia de la clase *blanda* y 1 de la clase *dura*. Por lo tanto,

$$\begin{aligned}(|E_{joven}|/|E|) \cdot Entropía(E_{joven}) &= \\&= (8/24) \cdot [-(4/8)\log_2(4/8) - (2/8)\log_2(2/8) - (2/8)\log_2(2/8)] = 0,495 \\(|E_{pre-presbicia}|/|E|) \cdot Entropía(E_{pre-presbicia}) &= \\&= (8/24) \cdot [-(5/8)\log_2(5/8) - (2/8)\log_2(2/8) - (1/8)\log_2(1/8)] = 0,426 \\(|E_{presbicia}|/|E|) \cdot Entropía(E_{presbicia}) &= \\&= (8/24) \cdot [-(6/8)\log_2(6/8) - (1/8)\log_2(1/8) - (1/8)\log_2(1/8)] = 0,347\end{aligned}$$

Luego, la ganancia de información con respecto al atributo *edad* es

$$Gain(E, edad) = 1,326 - (0,495 + 0,426 + 0,347) = 0,058 \quad (1)$$

- **Diagnóstico:** $E_{miope} = \{1, 2, 3, 4, 9, 10, 11, 12, 17, 18, 19, 20\}$ y $E_{hipermétrope} = \{5, 6, 7, 8, 13, 14, 15, 16, 21, 22, 23, 24\}$. En E_{miope} hay 7 instancias de la clase *ninguna*, 2 instancias de la clase *blanda* y 3 de la clase *dura*. De igual forma, en $E_{hipermétrope}$ hay 8 instancias de la clase *ninguna*, 3 instancias de la clase *blanda* y 1 de la clase *dura*. Por lo tanto,

$$\begin{aligned}(|E_{miope}|/|E|) \cdot Entropía(E_{miope}) &= \\&= (12/24) \cdot [-(7/12)\log_2(7/12) - (2/12)\log_2(2/12) - (3/12)\log_2(3/12)] = 0,6925 \\(|E_{hipermétrope}|/|E|) \cdot Entropía(E_{hipermétrope}) &= \\&= (12/24) \cdot [-(8/12)\log_2(8/12) - (3/12)\log_2(3/12) - (1/12)\log_2(1/12)] = 0,5945\end{aligned}$$

Luego, la ganancia de información con respecto al atributo *diagnóstico* es

$$Gain(E, diagnóstico) = 1,326 - (0,6925 + 0,5945) = 0,039 \quad (2)$$

- **Astigmatismo:** $E_{no} = \{1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22\}$ y $E_{si} = \{3, 4, 7, 8, 11, 12, 15, 16, 19, 20, 23, 24\}$. En E_{no} hay 7 instancias de la clase *ninguna*, 5 instancias de la clase *blanda* y 0 de la clase *dura*. De igual forma, en E_{si} hay 8

instancias de la clase *ninguna*, 0 instancias de la clase *blanda* y 4 de la clase *dura*. Por lo tanto,

$$\begin{aligned} (|E_{no}|/|E|) \cdot Entropía(E_{no}) &= \\ &= (12/24) \cdot [-(7/12)\log_2(7/12) - (5/12)\log_2(5/12) - (0/12)\log_2(0/12)] = 0,49 \\ (|E_{si}|/|E|) \cdot Entropía(E_{si}) &= \\ &= (12/24) \cdot [-(8/12)\log_2(8/12) - (0/12)\log_2(0/12) - (4/12)\log_2(4/12)] = 0,459 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *astigmatismo* es

$$Gain(E, \text{astigmatismo}) = 1,326 - (0,49 + 0,459) = 0,377 \quad (3)$$

- **Lágrimas:** $E_{reducida} = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23\}$ y $E_{normal} = \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}$. En $E_{reducida}$ hay 12 instancias de la clase *ninguna*, 0 instancias de la clase *blanda* y 0 de la clase *dura*. De igual forma, en E_{normal} hay 3 instancias de la clase *ninguna*, 5 instancias de la clase *blanda* y 4 de la clase *dura*. Por lo tanto,

$$\begin{aligned} (|E_{reducida}|/|E|) \cdot Entropía(E_{reducida}) &= \\ &= (12/24) \cdot [-(12/12)\log_2(12/12) - (0/12)\log_2(0/12) - (0/12)\log_2(0/12)] = 0 \\ (|E_{normal}|/|E|) \cdot Entropía(E_{normal}) &= \\ &= (12/24) \cdot [-(3/12)\log_2(3/12) - (5/12)\log_2(5/12) - (4/12)\log_2(4/12)] = 0,777 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *lágrimas* es

$$Gain(E, \text{lágrimas}) = 1,326 - (0 + 0,777) = 0,549 \quad (4)$$

De acuerdo con los valores (1), (2), (3) y (4), el atributo seleccionado para partir el nodo raíz es *lágrimas* ya que con él se obtiene la mayor ganancia de información. De momento, el árbol de decisión mostrado en la Figura 17.8 consta de dos nodos (además de la raíz), siendo el de la izquierda una hoja, ya que todas las instancias en ese nodo ($\{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23\}$) son de la clase *ninguna*.

Seguimos la construcción del árbol a partir del nodo de la derecha. Para ello, primero actualizamos E eliminando las instancias ya clasificadas, es decir, $E = \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}$. Volvemos a proceder como en el nodo raíz, pero ahora sólo tenemos que considerar los atributos *edad*, *diagnóstico* y *astigmatismo*.

Primer nivel-Nodo derecha: Primero calculamos las probabilidades de cada clase, sabiendo que de los 12 ejemplos de E hay 3 de clase *ninguna*, 5 de clase *blanda* y 4 de clase *dura*,

$$p_{ninguna} = 3/12 = 0,25 \quad p_{blanda} = 5/12 = 0,417 \quad p_{dura} = 4/12 = 0,33$$

y la entropía de la evidencia inicial E es

$$\begin{aligned} Entropía(E) &= -(3/12)\log_2(3/12) - (5/12)\log_2(5/12) - (4/12)\log_2(4/12) \\ &= 0,5 + 0,526 + 0,528 = 1,554 \end{aligned}$$

Dividimos la evidencia de acuerdo a los valores de los atributos y calculamos el segundo término de la Ecuación 17.2:

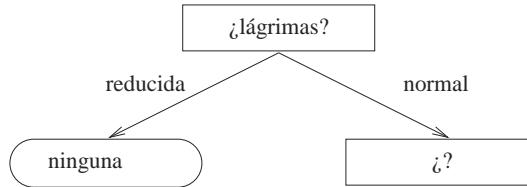


Figura 17.8: Primera partición en el árbol de decisión para el problema de las lentes de contacto.

- **Edad:** $E_{joven} = \{2, 4, 6, 8\}$, $E_{pre-presbicia} = \{10, 12, 14, 16\}$ y $E_{presbicia} = \{18, 20, 22, 24\}$. En E_{joven} hay 0 instancias de la clase *ninguna*, 2 instancias de la clase *blanda* y 2 de la clase *dura*. Análogamente, en $E_{pre-presbicia}$ hay 1 instancia de la clase *ninguna*, 2 instancias de la clase *blanda* y 1 de la clase *dura*; y, finalmente, en $E_{presbicia}$ hay 2 instancias de la clase *ninguna*, 1 instancia de la clase *blanda* y 1 de la clase *dura*. Por lo tanto,

$$\begin{aligned}
 & (|E_{joven}|/|E|) \cdot \text{Entropía}(E_{joven}) = \\
 & = (4/12) \cdot [-(0/4)\log_2(0/4) - (2/4)\log_2(2/4) - (2/4)\log_2(2/4)] = 0,33 \\
 & (|E_{pre-presbicia}|/|E|) \cdot \text{Entropía}(E_{pre-presbicia}) = \\
 & = (4/12) \cdot [-(1/4)\log_2(1/4) - (2/4)\log_2(2/4) - (1/4)\log_2(1/4)] = 0,495 \\
 & (|E_{presbicia}|/|E|) \cdot \text{Entropía}(E_{presbicia}) = \\
 & = (4/12) \cdot [-(1/4)\log_2(1/4) - (2/4)\log_2(2/4) - (1/4)\log_2(1/4)] = 0,495
 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *edad* es

$$Gain(E, \text{edad}) = 1,554 - (0,33 + 0,495 + 0,495) = 0,234 \quad (5)$$

- **Diagnóstico:** $E_{miope} = \{2, 4, 10, 12, 18, 20\}$ y $E_{hipermétrope} = \{6, 8, 14, 16, 22, 24\}$. En E_{miope} hay 1 instancia de la clase *ninguna*, 2 instancias de la clase *blanda* y 3 de la clase *dura*. De igual forma, en $E_{hipermétrope}$ hay 2 instancias de la clase *ninguna*, 3 instancias de la clase *blanda* y 1 de la clase *dura*. Por lo tanto,

$$\begin{aligned}
 & (|E_{miope}|/|E|) \cdot \text{Entropía}(E_{miope}) = \\
 & = (6/12) \cdot [-(1/6)\log_2(1/6) - (2/6)\log_2(2/6) - (3/6)\log_2(3/6)] = 0,7295 \\
 & (|E_{hipermétrope}|/|E|) \cdot \text{Entropía}(E_{hipermétrope}) = \\
 & = (6/12) \cdot [-(2/6)\log_2(2/6) - (3/6)\log_2(3/6) - (1/6)\log_2(1/6)] = 0,7295
 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *diagnóstico* es

$$Gain(E, \text{diagnóstico}) = 1,554 - (0,7295 + 0,7295) = 0,095 \quad (6)$$

- **Astigmatismo:** $E_{no} = \{2, 6, 10, 14, 18, 22\}$ y $E_{si} = \{4, 8, 12, 16, 20, 24\}$. En E_{no} hay 1 instancia de la clase *ninguna*, 5 instancias de la clase *blanda* y 0 de la clase *dura*. De igual forma, en E_{si} hay 2 instancias de la clase *ninguna*, 0 instancias

de la clase *blanda* y 4 de la clase *dura*. Por lo tanto,

$$\begin{aligned} (|E_{no}|/|E|) \cdot \text{Entropía}(E_{no}) &= \\ &= (6/12) \cdot [-(1/6)\log_2(1/6) - (5/6)\log_2(5/6) - (0/6)\log_2(0/6)] = 0,325 \\ (|E_{si}|/|E|) \cdot \text{Entropía}(E_{si}) &= \\ &= (6/12) \cdot [-(2/6)\log_2(2/6) - (0/6)\log_2(0/6) - (4/6)\log_2(4/6)] = 0,459 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *astigmatismo* es

$$Gain(E, \text{astigmatismo}) = 1,554 - (0,325 + 0,459) = 0,77 \quad (7)$$

De acuerdo con los valores (5), (6) y (7), el atributo seleccionado para partir el nodo derecho del primer nivel es *astigmatismo*. La Figura 17.9 muestra el árbol tras esta segunda partición.

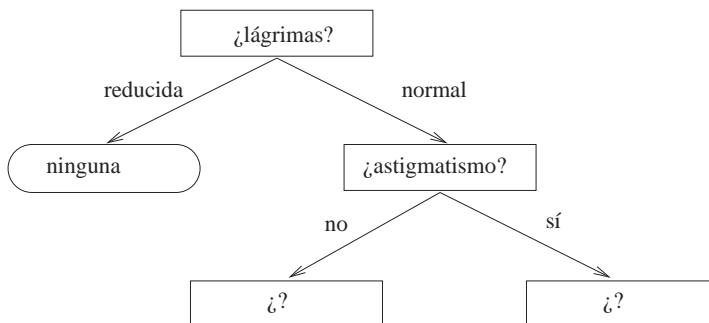


Figura 17.9: Segunda partición en el árbol de decisión para el problema de las lentes de contacto.

La construcción del árbol debe continuar explotando los nodos del segundo nivel ya que ninguno de ellos es puro. Llamemos N_i al nodo de la izquierda y N_d al nodo de la derecha.

Segundo nivel-Nodo izquierda N_i : Actualizamos la evidencia en N_i considerando únicamente los ejemplos en ese nodo, es decir, $E_i = \{2, 6, 10, 14, 18, 22\}$, de los cuales 5 son de clase *blanda* y 1 de clase *ninguna*. Esto nos permite calcular las probabilidades de las clases:

$$p_{ninguna} = 1/6 = 0,17 \quad p_{blanda} = 5/6 = 0,83 \quad p_{dura} = 0/6 = 0$$

y la entropía de E_i :

$$\begin{aligned} \text{Entropía}(E_i) &= -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) - (0/6)\log_2(0/6) \\ &= 0,43 + 0,219 + 0 = 1 \end{aligned}$$

A continuación pasamos a analizar los atributos *edad* y *diagnóstico*.

- **Edad:** $E_{i,joven} = \{2, 4\}$, $E_{i,pre-presbicia} = \{10, 14\}$ y $E_{i,presbicia} = \{18, 22\}$. Los dos ejemplos en $E_{i,joven}$ y $E_{i,pre-presbicia}$ son de la clase *blanda*, mientras que en $E_{i,presbicia}$ hay 1 ejemplo de la clase *ninguna* y 1 de la clase *blanda*. Por lo tanto,

$$\begin{aligned} (|E_{i,joven}|/|E|) \cdot Entropía(E_{i,joven}) &= \\ &= (2/6) \cdot [-(0/2)\log_2(0/2) - (2/2)\log_2(2/2) - (0/2)\log_2(0/2)] = 0 \\ (|E_{i,pre-presbicia}|/|E|) \cdot Entropía(E_{i,pre-presbicia}) &= \\ &= (2/6) \cdot [-(0/2)\log_2(0/2) - (2/2)\log_2(2/2) - (0/2)\log_2(0/2)] = 0 \\ (|E_{i,presbicia}|/|E|) \cdot Entropía(E_{i,presbicia}) &= \\ &= (2/6) \cdot [-(1/2)\log_2(1/2) - (1/2)\log_2(1/2) - (0/2)\log_2(0/2)] = 0,33 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *edad* es

$$Gain(E_i, edad) = 1 - (0 + 0 + 0,33) = 0,67 \quad (8)$$

- **Diagnóstico:** $E_{i,miope} = \{2, 10, 18\}$ y $E_{i,hipermetrope} = \{6, 14, 22\}$. En $E_{i,miope}$ hay 1 instancia de la clase *ninguna* y 2 instancias de la clase *blanda*. De igual forma, en $E_{i,hiperméetrope}$ todas las instancias son de la clase *blanda*. Por lo tanto,

$$\begin{aligned} (|E_{i,miope}|/|E|) \cdot Entropía(E_{i,miope}) &= \\ &= (3/6) \cdot [-(1/3)\log_2(1/3) - (2/3)\log_2(2/3) - (0/3)\log_2(0/3)] = 0,459 \\ (|E_{i,hiperméetrope}|/|E|) \cdot Entropía(E_{i,hiperméetrope}) &= \\ &= (3/6) \cdot [-(0/3)\log_2(0/3) - (3/3)\log_2(3/3) - (0/3)\log_2(0/3)] = 0 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *diagnóstico* es

$$Gain(E, diagnóstico) = 1 - (0,459 + 0) = 0,541 \quad (9)$$

Como (8) es mayor que (9) el atributo seleccionado es *edad* cuyos hijos son dos de ellos puros y por lo tanto hojas, mientras que el tercero se parte en dos usando el atributo *diagnóstico* que es el único que podemos seleccionar en ese nodo. La Figura 17.10 muestra el árbol hasta este punto.

Segundo nivel-Nodo derecha N_d : Vamos a proceder de forma similar al caso anterior. La evidencia en N_d es $E_d = \{4, 8, 12, 16, 20, 24\}$, 2 de cuyas instancias son de clase *ninguna* y 4 de clase *dura*. Esto nos permite calcular las probabilidades de las clases:

$$p_{ninguna} = 2/6 = 0,33 \quad p_{blanda} = 0/6 = 0 \quad p_{dura} = 4/6 = 0,67$$

y la entropía de E_d :

$$\begin{aligned} Entropía(E_d) &= -(2/6)\log_2(2/6) - (0/6)\log_2(0/6) - (4/6)\log_2(4/6) \\ &= 0,528 + 0 + 0,39 = 0,918 \end{aligned}$$

A continuación pasamos a analizar los atributos *edad* y *diagnóstico*.

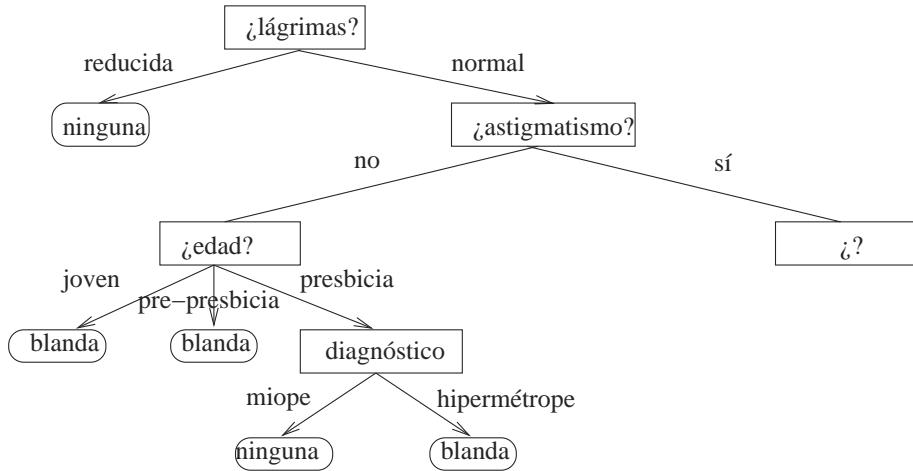


Figura 17.10: Tercera y cuarta partición en el árbol de decisión para el problema de las lentes de contacto.

- **Edad:** $E_{d,joven} = \{4, 8\}$, $E_{d,pre-presbicia} = \{12, 16\}$ y $E_{d,presbicia} = \{20, 24\}$. Los dos ejemplos en $E_{d,joven}$ son de clase *dura*, mientras que en $E_{d,pre-presbicia}$ y $E_{d,presbicia}$ hay 1 ejemplo de la clase *ninguna* y 1 de la clase *dura*. Por lo tanto,

$$\begin{aligned}
 &(|E_{d,joven}|/|E|) \cdot \text{Entropía}(E_{d,joven}) = \\
 &\quad = (2/6) \cdot [-(0/2)\log_2(0/2) - (0/2)\log_2(0/2) - (2/2)\log_2(2/2)] = 0 \\
 &(|E_{d,pre-presbicia}|/|E|) \cdot \text{Entropía}(E_{d,pre-presbicia}) = \\
 &\quad = (2/6) \cdot [-(1/2)\log_2(1/2) - (0/2)\log_2(0/2) - (1/2)\log_2(1/2)] = 0,33 \\
 &(|E_{d,presbicia}|/|E|) \cdot \text{Entropía}(E_{d,presbicia}) = \\
 &\quad = (2/6) \cdot [-(1/2)\log_2(1/2) - (0/2)\log_2(0/2) - (1/2)\log_2(1/2)] = 0,33
 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *edad* es

$$Gain(E_d, edad) = 1 - (0,33 + 0 + 0,33) = 0,34 \quad (10)$$

- **Diagnóstico:** $E_{d,miope} = \{4, 12, 20\}$ y $E_{d,hipermétrope} = \{8, 16, 24\}$. Los tres ejemplos de $E_{d,miope}$ son de la clase *dura*, y en $E_{d,hipermétrope}$ hay 2 instancias de la clase *ninguna* y 1 instancia de la clase *dura*. Por lo tanto,

$$\begin{aligned}
 &(|E_{d,miope}|/|E|) \cdot \text{Entropía}(E_{d,miope}) = \\
 &\quad = (3/6) \cdot [-(0/3)\log_2(0/3) - (0/3)\log_2(0/3) - (3/3)\log_2(3/3)] = 0 \\
 &(|E_{d,hipermétrope}|/|E|) \cdot \text{Entropía}(E_{d,hipermétrope}) = \\
 &\quad = (3/6) \cdot [-(2/3)\log_2(2/3) - (0/3)\log_2(0/3) - (1/3)\log_2(1/3)] = 0,98
 \end{aligned}$$

Luego, la ganancia de información con respecto al atributo *diagnóstico* es

$$Gain(E, diagnóstico) = 1 - (0 + 0,98) = 0,02 \quad (11)$$

Nuevamente, el atributo seleccionado es *edad*. Uno de sus nodos hijo es una hoja (de clase *dura*) mientras que los otros dos nodos deben volver a partirse usando en ambos casos el atributo *diagnóstico*. La Figura 17.11 muestra el árbol de decisión completamente generado.

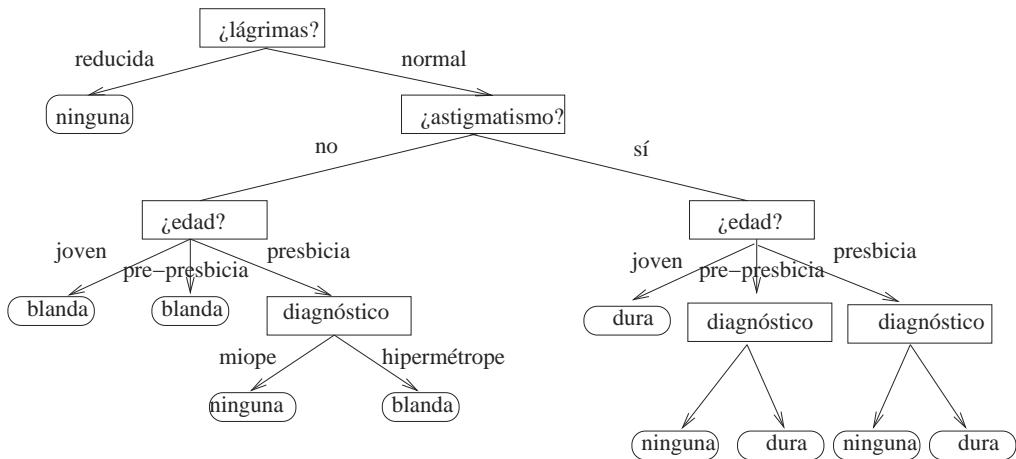


Figura 17.11: Árbol de decisión para el problema de las lentes de contacto.

■

17.11 Ejercicios propuestos

17.1. Considerar el problema de clasificar animales cuya evidencia se muestra en la Tabla 17.4³.

nº	Animal	Cobertura	Habitat	Patas	Lactancia	HT	Oviparo	Branq.	Clase
1	perro	pelo	tierra	4	sí	sí	no	no	mamif.
2	delfín	ninguna	agua	0	sí	sí	no	no	mamif.
3	murcielago	pelo	aire	2	sí	sí	no	no	mamif.
4	ornitorrinco	pelo	agua	2	sí	sí	sí	no	mamif.
5	trucha	escamas	agua	0	no	no	sí	sí	pez
6	arenque	escamas	agua	0	no	no	sí	sí	pez
7	tiburón	ninguna	agua	0	no	no	sí	sí	pez
8	anguila	ninguna	agua	0	no	no	sí	sí	pez
9	lagarto	escamas	tierra	4	no	no	sí	no	reptil
10	cocodrilo	escamas	tierra	4	no	no	sí	no	reptil
11	tiranosaurio-rex	tierra	si	4	no	no	sí	no	reptil
12	serpiente	escamas	tierra	0	no	no	sí	no	reptil
13	tortuga	escamas	agua	4	no	no	sí	no	reptil
14	aguila	plumas	aire	2	no	sí	sí	no	ave
15	avestruz	plumas	tierra	2	no	sí	sí	no	ave
16	pingüino	plumas	agua	2	no	sí	sí	no	ave

Tabla 17.4: Evidencia del problema de clasificar animales.

³HT = Homeotérmico y Branq. = Branquias

1. Utilizar el criterio de ganancia de información para seleccionar el mejor atributo para ser el nodo raíz de un árbol de decisión para este problema.
2. Indica cómo se parte la evidencia de acuerdo al atributo seleccionado.

17.2. De acuerdo al Algoritmo 17.1 de inducción de árboles de decisión, ¿es posible que en un mismo árbol haya dos nodos diferentes con el mismo test? Justifica la respuesta.

17.3. Una entidad bancaria ha decidido aplicar aprendizaje automático para que le ayude a determinar si concede o no un crédito. Para ello, dispone de la evidencia de la Tabla 17.5. Se decide utilizar una técnica de inducción de árboles de decisión, generando el modelo de la Figura 17.12. Convertir este árbol de decisión en un árbol PET. Crear tres versiones dependiendo de la manera de calcular la probabilidad: sin correcciones, con la corrección de Laplace, y con la corrección m-estimado.

Edad	Sueldo	Casa-propiedad	Segunda-vivienda	Estado-civil	Crédito
24	50	no	no	soltero	denegar
34	90	no	no	casado	denegar
45	125	sí	no	casado	denegar
60	60	no	no	casado	denegar
31	200	no	no	casado	denegar
29	140	no	no	soltero	denegar
50	250	sí	sí	soltero	conceder
41	320	sí	no	casado	conceder
30	670	no	no	casado	conceder
35	400	sí	no	soltero	conceder
47	360	sí	no	casado	conceder
42	310	sí	no	soltero	conceder

Tabla 17.5: Conjunto de entrenamiento.

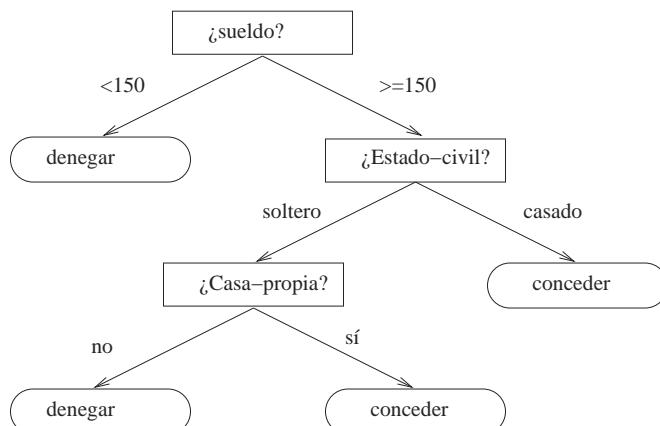


Figura 17.12: Árbol de decisión para el problema de la entidad bancaria.

17.4. La precisión de un árbol de decisión es el porcentaje de ejemplos del conjunto de test que clasifica correctamente. Calcular la precisión del árbol del ejercicio anterior usando el conjunto de test la Tabla 17.6.

Edad	Sueldo	Casa-propiedad	Segunda-vivienda	Estado-civil	Crédito
22	200	sí	sí	soltero	denegar
30	90	sí	no	casado	denegar
36	125	sí	no	soltero	denegar
40	60	no	no	casado	denegar
33	175	no	no	soltero	conceder
31	210	no	no	casado	conceder
47	145	sí	sí	casado	conceder
51	300	sí	no	casado	conceder

Tabla 17.6: Conjunto de test.

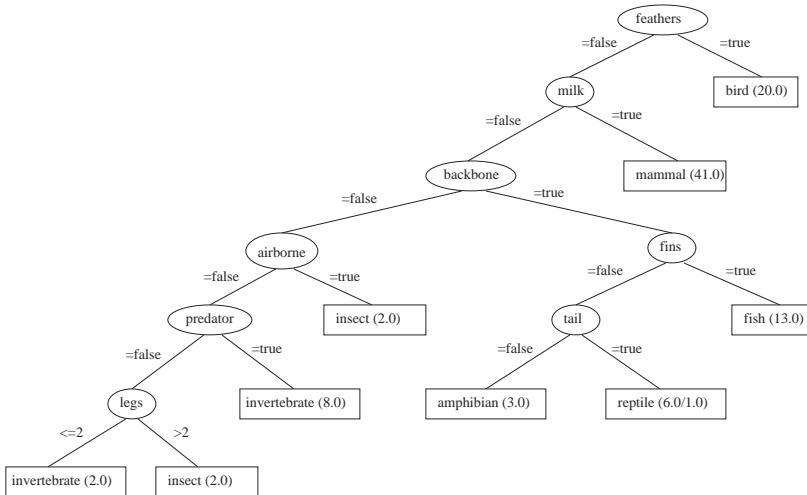


Figura 17.13: Árbol de decisión para el problema *zoo*.

17.5. Utilizar Weka para calcular diferentes modelos basados en reglas. Usar el dataset del UCI *Iris*, y aplicar los métodos C4.5 de árboles de decisión y PRISM. Comparar los modelos.

17.6. Expresar el árbol de decisión de la Figura 17.13, obtenido con el algoritmo J4.8 del sistema Weka para el problema *zoo* del repositorio UCI, como un conjunto de reglas.

Referencias

- BARRON, A. R.; RISSANEN, J. y YU, B.: «The Minimum Description Length Principle in Coding and Modeling.» *IEEE Transactions on Information Theory*, 1998, **44**(6), pp. 2743–.
- BAUER, E. y KOHAVI, R.: «An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants». *Machine Learning*, 1999, **36**, p. 105.
- BREIMAN, L.: «Bagging Predictors». *Machine Learning*, 1996, **24**(2), pp. 123–140.
- BREIMAN, L.; FRIEDMAN, J.; OLSHEN, R. y STONE, C.: *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
- BRESLOW, L. A. y AHA, D. W.: «Simplifying decision trees: A survey». *Knowl. Eng. Rev.*, 1997, **12**(1), pp. 1–40. ISSN 0269-8889.
- BRODLEY, C. E. y UTGOFF, P. E.: «Multivariate Decision Trees». *Machine Learning*, 1995, **19**, p. 45.
- BUNTINE, W. y NIBLETT, T.: «A Further Comparison of Splitting Rules for Decision-Tree Induction». *Machine Learning*, 1992, **8**(1), pp. 75–85.
- CENDROWSKA, J.: «PRISM: an algorithm for inducing modular rules». *Journal of Man-Machine Studies*, 1987, **27**, pp. 349–370.
- CESTNIK, B.; KONONENKO, I. y BRATKO, I.: «ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users.» En: *Proceedings of EWSL 87: 2nd European Working Session on Learning*, pp. 31–45, 1987.
- CLARK, P. y NIBLETT, T.: «The CN2 Induction Algorithm». *Machine Learning*, 1989, **3**, pp. 261–283.
- DIETTERICH, T. G.: «Ensemble Methods in Machine Learning». En: *Proceedings of the First International Workshop on Multiple Classifier Systems*, volumen 1857 de *Lecture Notes in Computer Science*. Springer, 2000a.
- DIETTERICH, T. G.: «An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization». *Machine Learning*, 2000b, **40**(2), p. 1.
- DOWE, D. L.; OLIVER, J. J. y WALLACE, C. S.: «Inferring Decision Graphs using the Minimum Message Length Principle». En: *5th Australian Joint Conference on Artificial Intelligence*, World Scientific, Singapore, 1992.
- ELOMAA, T. y ROUSU, J.: «Finding Optimal Multi-Splits for Numerical Attributes in Decision Tree Learning». *Informe técnico*, University of Helsinki, 1996.

- ESPOSITO, F.; MALERBA, D. y SEMERARO, G.: «A Comparative Analysis of Methods for Pruning Decision Trees». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997, **19(5)**, pp. 476–491.
citeseer.ist.psu.edu/esposito97comparative.html
- FAYYAD, U. M. y IRANI, K. B.: «On the Handling of Continuous-Valued Attributes in Decision Tree Generation.» *Machine Learning*, 1992, **8**, pp. 87–102.
- FERRI, C.; FLACH, P. A. y HERNÁNDEZ-ORALLO, J.: «Improving the AUC of Probabilistic Estimation Trees.» En: *Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, pp. 121–132, 2003.
- FRANK, E. y WITTEN, I. H.: «Generating accurate rule sets without global optimization». En: *Proc. 15th International Conf. on Machine Learning*, pp. 144–151. Morgan Kaufmann, San Francisco, CA, 1998.
- FREUND, Y. y SCHAPIRA, R. E.: «Experiments with a new Boosting algorithm». En: *Proc. 13th International Conference on Machine Learning*, pp. 148–146. Morgan Kaufmann, 1996.
- GAMA, J. y BRAZDIL, P.: «Cascade Generalization». *Machine Learning*, 2000, **41(3)**, p. 315.
- HAN, J. y KAMBER, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000. ISBN 1-55860-489-8.
- HERNÁNDEZ, J.; RAMÍREZ, M. J.; y FERRI, C.: *Introducción a la Minería de Datos*. Pearson Prentice Hall, 2004.
- HOLTE, R. C.: «Very Simple Classification Rules Perform Well on Most Commonly Used Datasets». *Machine Learning*, 1993, **11**, pp. 63–91.
- KUNCHEVA, L. I.: *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.
- LANDWEHR, N.; HALL, M. y FRANK, E.: «Logistic Model Trees». *Machine Learning*, 2005, **59(1-2)**, pp. 161–205.
- LIU, B.; XIA, Y. y YU, P. S.: «Clustering Through Decision Tree Construction». En: *Proceedings of the ninth international conference on Information and knowledge management*, pp. 20–29, 2000.
citeseer.ist.psu.edu/liu00clustering.html
- MEHTA, M.; RISSANEN, J. y AGRAWAL, R.: «MDL-Based Decision Tree Pruning». En: *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pp. 216–221, 1995.

MELVILLE, P. y MOONEY, R.: «Constructing diverse classifier ensembles using artificial training examples». En: *Proc. of 18th Intl. Joint Conf. on Artificial Intelligence*, pp. 505–510, 2003.

MICHALSKI, R. y LARSON, J.: «Incremental Generation of VL1 Hypotheses: the underlying Methodology and the Description of Program AQ11». *ISG 83-5*, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1980.

MICHALSKI, R.; MOZETIC, I.; HONG, J. y LAVRAC, N.: «The AQ15 Inductive Learning System: an Overview and Experiments». En: *Proceedings of IMAL 1986*, Université de Paris-Sud, Orsay, 1986.

MITCHELL, T. M.: *Machine Learning*. McGraw-Hill, New York, 1997.

MURTHY, S. K.: «Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey». *Data Mining and Knowledge Discovery*, 1998, **2**(4), pp. 345–389.

NEWMAN, D.J.; HETTICH, S.; BLAKE, C.L. y MERZ, C.J.: «UCI Repository of machine learning databases», 1998.

<http://www.ics.uci.edu/~mlearn/MLRepository.html>

PROVOST, F. J. y DOMINGOS, P.: «Tree Induction for Probability-Based Ranking.» *Machine Learning*, 2003, **52**(3), pp. 199–215.

QUINLAN, J. R.: «Learning efficient classification procedures and their application to chess end games». En: Ryszard S. Michalski; Jaime G. Carbonell y Tom M. Mitchell (Eds.), *Machine Learning, an Artificial Intelligence approach*, volumen 1, pp. 463–482. Morgan Kaufmann, San Mateo, California, 1983.

QUINLAN, J. R.: «Induction of Decision Trees.» *Machine Learning*, 1986, **1**(1), pp. 81–106.

QUINLAN, J. R.: «Simplifying Decision Trees.» *International Journal of Man-Machine Studies*, 1987, **27**(3), pp. 221–234.

QUINLAN, J. R.: «Learning with Continuous Classes». En: *5th Australian Joint Conference on Artificial Intelligence*, pp. 343–348, 1992.

QUINLAN, J. R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. ISBN 1558602380.

QUINLAN, J. R.: «Bagging, Boosting, and C4. 5». En: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 725–730. AAAI Press / MIT Press, Menlo Park. ISBN 0-262-51091-X, 1996.

UTGOFF, P. E.: «Perceptron Trees: A Case Study in Hybrid Concept Representations». *Connection Science*, 1989, **1**(4), pp. 377–391.

- UTGOFF, P. E.; BERKMAN, N. C. y CLOUSE, J. A.: «Decision Tree Induction Based on Efficient Tree Restructuring.» *Machine Learning*, 1997, **29**(1), pp. 5–44.
- WALLACE, C. S. y DOWE, D. L.: «Minimum Message Length and Kolmogorov Complexity». *The Computer Journal*, 1999, **42**(4), pp. 270–283.
- WITTEN, I. y FRANK, E.: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 2005.

Capítulo 18

Técnicas de extracción de reglas

María José Ramírez Quintana y José Hernández Orallo
Universidad Politécnica de Valencia

18.1 Introducción

Es un tópico referirse al aumento del volumen de información que nos invade como una de las causas y, por tanto, uno de los motores, de muchas de las tecnologías informáticas que han aparecido en los últimos años. Pero no suele aludirse tan frecuentemente el hecho de que, asociado a este crecimiento de volumen, se ha producido un aumento no menos considerable de la abstracción de la información que se procesa automáticamente. La gestión, procesamiento, intercambio de datos ha dado lugar a la gestión, procesamiento e intercambio de conocimiento.

Como vimos en los capítulos 2 al 7, existen diferentes formas de representación de conocimiento. Todas ellas, finalmente, parten o se pueden reducir a un conjunto de fórmulas lógicas (proposicionales, de primer orden, enriquecidas con algún tipo de expresión de incertidumbre o de modo) y finalmente, orientadas en forma de reglas de la forma si-entonces, donde la riqueza expresiva del antecedente y del consecuente marcan la riqueza expresiva del lenguaje de representación. Así, vimos en el capítulo 3 que los sistemas basados en reglas son una herramienta frecuente en la ingeniería del conocimiento y, en general, en IAI. Dado un sistema de reglas, sabemos extraer las consecuencias lógicas de ellas, sabemos establecer si existen inconsistencias, sabemos intercambiar las reglas entre diferentes sistemas, sabemos representar las reglas y disponerlas de una manera que sea comprensible para los seres humanos.

El problema es que no todo el conocimiento del que disponemos o que extraemos se encuentra en forma de reglas. Gran parte de este conocimiento ha sido generado manualmente por seres humanos y tiene una representación informal, generalmente en lenguaje natural, o ni siquiera dispone de una representación precisa, encontrándose en la mente de una o más personas. Otra parte, cada día más importante, de este conocimiento no explícito se encuentra en modelos extraídos automáticamente, pero que tienen una representación específica, frecuentemente en forma de complejas estructuras matemáticas, que difícilmente puede comprenderse o intercambiarse.

Del primer tipo de conocimiento, el que tiene un origen manual, hemos visto en el capítulo 3 que no resulta siempre fácil de expresar en la base de hechos y conocimientos de un sistema basado en reglas u otros tipos de sistemas expertos. La adquisición de conocimiento, como también veremos en el capítulo 19, sigue siendo un gran cuello de botella, porque los seres humanos nos basamos en mucho conocimiento tácito, incluso inconsciente, que es difícil explicitar.

El segundo tipo de conocimiento, el que tiene un origen automático, se genera por diferentes técnicas de aprendizaje o de extracción de conocimiento que estamos viendo en los capítulos del 15 al 18. El capítulo 15 demuestra de que la extracción de modelos a partir de datos utilizando técnicas sofisticadas de aprendizaje mediante datos, como por ejemplo las redes neuronales o las máquinas de vectores soporte, puede proporcionar modelos muy precisos y, por tanto, muy útiles, pero que son incomprensibles y difícilmente manejables o integrables con el conocimiento anterior.

Tanto el primer tipo de conocimiento, el tácito o informal, como el segundo, el formal, pero que se basa en estructuras complejas, dificulta una verdadera ingeniería del conocimiento que permita su intercambio, su integración y su mantenimiento.

En este capítulo, presentamos una serie de técnicas que permiten convertir distintos tipos de modelos en conjuntos de reglas. El hecho de que nos centremos en aquellas técnicas que producen conjuntos de reglas se debe a que los sistemas de reglas son más fáciles de entender por los humanos y, en segundo lugar, porque los conjuntos de reglas son más fáciles de integrar en sistemas expertos, ontologías y otros tipos de sistemas basados en conocimiento, y su dinámica e interrelación más fácil de analizar. Además, permiten homogeneizar en un mismo lenguaje de representación, modelos, sistemas o conocimiento expresados de manera heterogénea. Por todo ello, la mayoría de técnicas que intentan convertir modelos incomprensibles (o ambiguos) en modelos formales y comprensibles se basan en conjuntos de reglas.

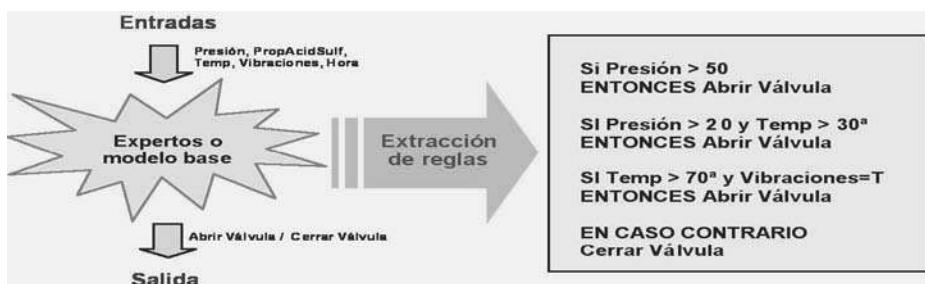


Figura 18.1: Gráfico con un ejemplo de extracción de reglas.

En la Figura 18.1, vemos cómo un grupo de expertos, ayudados con varios modelos matemáticos, deciden cuándo ha de ponerse en marcha una válvula de una central eléctrica. Este grupo, junto con todo su conocimiento (implícito y explícito) y las herramientas que usa funciona como un sistema complejo que toma una decisión

(una salida) a partir de unas variables (las entradas). Un proceso de extracción de reglas puede permitir representar el sistema complejo como una serie de reglas que relacionan explícita y comprensiblemente las entradas con la salida.

En general, cuando hablamos de “modelo”, nos referimos a cualquier sistema que, ante unas entradas, obtiene una salida (o más). En principio, podría creerse que, según la terminología del aprendizaje automático, esta noción abarca únicamente modelos de tipo predictivo (es decir, clasificadores y regresores), sin embargo también incluye algunos modelos de tipo descriptivo. Por ejemplo, un sistema de agrupamiento basado en una técnica de *clustering* incomprensible puede convertirse en un modelo de *clustering* comprensible, tomando los grupos como clases, es decir, como el valor de la salida.

Antes de pasar a abordar las diversas técnicas es importante precisar que no siempre cualquier conjunto de reglas es una buena solución. Una elaborada pero elegante ecuación matemática puede ser más precisa, más comprensible y más manejable que cientos de reglas. Por tanto, las técnicas que extraen reglas suelen buscar un compromiso entre fidelidad y comprensibilidad. La comprensibilidad de un modelo está generalmente ligada a la simplicidad del mismo, aunque es importante destacar que no son conceptos equivalentes. El grado de comprensibilidad de un modelo es un factor subjetivo, ya que depende en gran medida de la experiencia y conocimiento de los usuarios [Kohavi y Sommerfield, 1998; Kononenko, 1990; Pazzani, 1991]. Si nos restringimos a los modelos basados en reglas y buscando medidas objetivas, se suele tener exclusivamente en cuenta el tamaño de los modelos, determinado por las características del conjunto de reglas; por ejemplo, el número de reglas y la cantidad de condiciones por regla [Engelbrecht y otros, 2001; Lu y otros, 1995; Tickle y otros, 1998]. Por ejemplo, [Lu y otros, 1995] proponen una función que involucra el número de reglas extraídas (R) y el número total de condiciones (C) según la siguiente fórmula:

$$\text{Complejidad} = 2 \cdot R + C$$

Es decir, el número de reglas pesa el doble que el total de condiciones. Algunas de estas métricas se combinan con la medida de acierto, fidelidad, ajuste o precisión para tener una métrica conjunta de optimalidad, que evalúe el compromiso que hemos comentado anteriormente. Esta aproximación es muy similar a los clásicos principios de la longitud de descripción mínima (*Minimum Description Length*, MDL) [Li y Vitányi, 1997; Rissanen, 1978] o el principio de la longitud mínima de mensaje (*Minimum Message Length*, MML)¹ [Wallace y Boulton, 1968], que si bien es la aproximación con base más robusta, siempre aparece la arbitrariedad de la codificación que equivale, en cierto modo, a la arbitrariedad de la ponderación de ajuste y simplicidad de otras aproximaciones.

En definitiva, en lo que resta de este capítulo veremos diferentes técnicas que intentan extraer modelos comprensibles a partir de modelos no comprensibles, buscando

¹Ambos criterios difieren ligeramente [Wallace y Dowe, 1999], pero la determinación del coste de representación de un modelo de clasificación y los datos que no pueden ser clasificados por él son similares y podemos usar ambos criterios de manera indistinta.

este compromiso entre fidelidad y simplicidad. Estas técnicas tienen aplicaciones múltiples, no sólo para la adquisición de conocimiento, sino también para la ingeniería inversa, para la integración de conocimiento, para la interacción persona-computador y muchas otras.

El capítulo se estructura como sigue. En primer lugar, veremos las técnicas que son capaces de extraer reglas independientemente del tipo de modelo, técnicas que denominamos de caja negra, porque no necesitan examinar las interioridades de los modelos o expertos a imitar y que pueden utilizarse para extraer reglas de sistemas tan variados como un ser humano, un sistema experto, una red neuronal, un multiclásificador, etc. En segundo lugar, abordaremos las técnicas que se especializan en diferentes tipos de modelos base. Así, veremos las técnicas de extracción de reglas específicas sobre redes neuronales y sobre máquinas de vectores soporte. Finalmente, repasaremos las técnicas que extraen otros tipos de reglas, en especial las técnicas de extracción de reglas borrosas.

18.2 Técnicas de extracción de reglas a partir de modelos de caja negra

Como hemos dicho en la introducción, dado un modelo existente, con cualquier representación implícita o explícita, el objetivo es convertirlo en un modelo (aproximadamente) equivalente, que sea más comprensible que el modelo original. Para ello, como también hemos comentado, existen dos aproximaciones principales. En la primera aproximación, podemos utilizar la estructura interna del modelo con el fin de explicitarlo. En la segunda aproximación, tratamos el modelo original como si fuera una caja negra. La gran ventaja de la primera aproximación es que al acceder al interior de la caja, la traducción a un modelo comprensible puede “reutilizar” parte de las estructuras o patrones existentes en el modelo original y explicitarlos. La gran desventaja de la primera aproximación es que necesitamos “traductores” para cada tipo específico de modelo original, ya que dependen de la estructura específica de cada sistema. De este modo, necesitamos técnicas específicas para convertir redes neuronales en reglas, técnicas específicas para convertir máquinas de vectores soporte en reglas, técnicas específicas para extraer reglas de un experto humano, etc. Esta desventaja de la primera aproximación es precisamente la gran ventaja de la segunda aproximación: las técnicas de extracción de reglas de caja negra pueden funcionar independientemente del tipo de modelo original. Dicho de otra manera, las técnicas de extracción de reglas de caja negra sólo necesitan la semántica del modelo original, generalmente en forma de casos o conjuntos de ejemplos de entradas y salidas, y no de su sintaxis o de su funcionamiento interno.

Las técnicas de extracción de reglas a partir de modelos de caja negra se desarrollaron inicialmente para la adquisición de conocimiento en sistemas expertos. Como veremos en el capítulo 19, esta adquisición es el cuello de botella de la creación de sistemas expertos [Brulé y Bount, 1998; Debenham, 1998; Feigenbaum, 2003; Greenwell, 1988; Meyer y Booker, 2001]. Si la adquisición de conocimiento se realiza manualmente, muchos expertos no pueden expresar su conocimiento en forma de reglas claras

e inequívocas. Los expertos emplean generalmente reglas explícitas, reglas “a ojo” y “reglas instintivas con parte subconsciente”. Incluso en el caso de que el experto pueda escribir todo su conocimiento, esto representa un alto esfuerzo, puede ser un proceso donde se desperdicia mucho tiempo, es difícil de mantener y el resultado es, a veces, un modelo trascrito que no se puede aplicar de una forma completamente automatizada puesto que todavía hay una cierta ambigüedad.

En este contexto y, paralelamente al desarrollo de técnicas de aprendizaje automático que generan reglas, aparecieron diferentes métodos de adquisición de conocimiento basados en técnicas de aprendizaje automático (por ejemplo: [Grover, 1983], *BGM* [Martínez, 1994]). En general, podemos resumir los pasos que siguen estos métodos en: (1) la definición del dominio del problema, en donde se recopilan casos de uso y luego uno o varios expertos “responden” a ejemplos, actuando como si fueran un “oráculo”; (2) la formulación del conocimiento, aquí es donde se aplican las técnicas de aprendizaje automático para obtener los prototipos; (3) la verificación del conocimiento obtenido.

Este tipo de técnicas de extracción de reglas a partir de modelos de caja negra también se desarrollaron para la extracción de reglas a partir de redes neuronales, aunque las técnicas fueron analizadas de manera independiente. En el contexto de las redes neuronales, se denominan generalmente “técnicas pedagógicas” [Andrews y otros, 1995], en contraposición a otras técnicas que sí que indagan en la estructura del sistema. Ambas técnicas se estudiarán en la siguiente sección.

Dentro del área del aprendizaje automático, la misma idea recurrente aparece para extraer modelos simples a partir de la combinación de árboles de decisión: usando *arcing* [Breiman y Shang, 1997], *bagging* [Domingos, 1997a] o *boosting* [Estruch y otros, 2003] (véase la sección 17.6 y el capítulo 17). Otras aproximaciones extraen reglas a partir de la combinación de redes neuronales artificiales [Zhou y Jiang, 2003, 2004; Zhou y otros, 2003].

Todos estos trabajos sugieren el paradigma de aprender dos veces para desarrollar modelos precisos y comprensibles, es decir, una primera etapa de modelización que tiene como objetivo conseguir una alta precisión, y una segunda etapa en la que se persigue la comprensibilidad, como se ve en la Figura 18.2. Sin embargo, hasta hace poco, no se ha estudiado de manera genérica las características de esta extracción de cualquier modelo, independientemente de su contenido.

El método general de dos fases puede ilustrarse utilizando, como base, el método denominado *combinación de múltiples modelos* (CMM) [Domingos, 1997a,b, 1998]. La principal idea del método CMM es recuperar la pérdida de comprensibilidad por la combinación de múltiples modelos. Aunque inicialmente ideado para multiclassificadores, la idea es muy simple y aplicable a cualquier modelo de origen. Siguiendo con la Figura 18.2, en la primera etapa se genera un modelo (que podemos llamar oráculo) usando un conjunto de entrenamiento. En la segunda etapa, simplemente se genera un conjunto de datos inventados que se etiquetan con el modelo existente. Este nuevo conjunto de datos junto con el conjunto original de entrenamiento se utiliza para entrenar un modelo basado en reglas, por ejemplo, un árbol de decisión. El modelo aprendido se puede denominar modelo “mímético”, porque, en cierto modo, imita o mimetiza el modelo original.

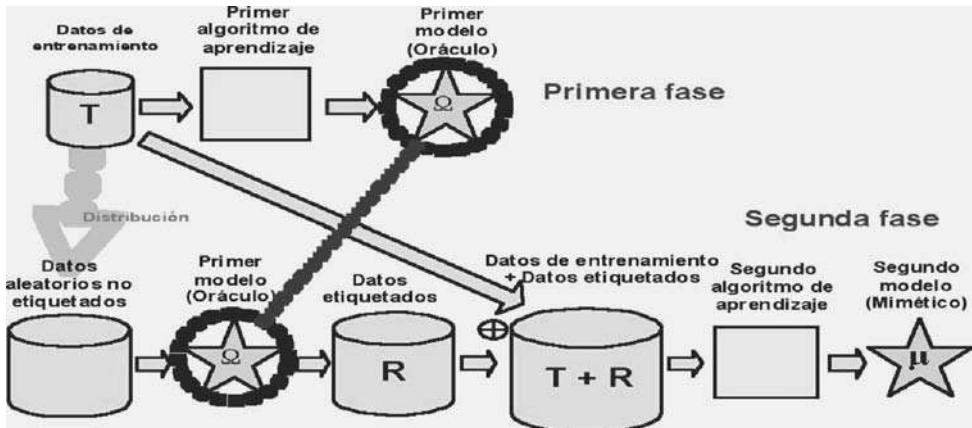


Figura 18.2: Aprendizaje mimético con datos de entrenamiento originales.

Lógicamente, el modelo mimético es menos preciso que el modelo original, especialmente si el poder de representación del sistema original (el oráculo) es mayor que el del método usado para obtener el modelo mimético. Sin embargo, y de forma general, se sabe que la precisión y estabilidad de un modelo se incrementa con el tamaño del conjunto de entrenamiento (debido a la disminución de la varianza [Kohavi y Wolpert, 1996]). En el caso de la generación del modelo mimético, el tamaño del conjunto de entrenamiento puede ser tan grande como se quiera, ya que en parte es inventado, con lo que, en principio, podemos ajustarnos al modelo original en gran medida.

Según Domingos, dos puntos significativos se deben tener en cuenta en el método CMM. El primero es que los ejemplos inventados deben ser generados de la misma distribución original (“verdadera”) y no según la distribución uniforme. El segundo es que es conveniente incluir los datos originales de entrenamiento en la generación del modelo mimético, como se ve en la figura anterior.

El trabajo experimental de Domingos se realizó sobre 26 conjuntos del repositorio UCI. El modelo combinado se obtuvo aplicando *bagging*, mientras que como algoritmo mimético se usó C4.5Rules. Los resultados obtenidos demostraron que el segundo modelo tenía una precisión cercana al 60 % de la ganancia de la precisión del modelo combinado mientras que su complejidad en términos del número de reglas del modelo era entre 2 y 6 veces mayor que un modelo simple generado usando C4.5Rules a partir de únicamente los datos originales. Más recientemente, en [Estruch y otros, 2003] se analizó experimentalmente el método mimético pero con un modelo combinado diferente al usado en la aproximación CMM. En concreto se aplicó *boosting*, obteniéndose unos resultados consistentes con los de Domingos.

En diversos estudios resumidos en [Blanco-Vega, 2007] se analiza todo el proceso de manera general, denominando “técnica mimética” a la técnica general, y se desarrollan variantes y nuevas aplicaciones. En primer lugar, se diferencia entre la aplicación de la técnica mimética usando los datos de entrenamiento originales (aplicable para

comprender una red neuronal, un multiclásificador, una máquina de vectores soporte recientemente aprendido) y la técnica sin el uso de estos datos (aplicable a un abanico mayor de situaciones, incluidas la adquisición de conocimiento a partir de expertos, la ingeniería inversa ...).

El esquema general tiene su punto más importante, como ya destacó Domingos, en la generación de un conjunto de datos inventados. Este conjunto es el que posteriormente es etiquetado por el oráculo y, a partir de él (junto con algunos otros datos de entrenamiento originales si se dispone de ellos) se entrena el modelo mimético. Por tanto, es fundamental determinar cómo se van a generar los datos inventados, siendo deseable que se ajusten a la distribución original y que sean exhaustivos (que puedan cubrir todas las posibilidades). Al igual que en el aprendizaje semisupervisado, existen diferentes maneras de generar datos inventados:

- Distribución uniforme: si disponemos de los valores posibles para los atributos nominales y del valor máximo y mínimo para los atributos numéricos, la manera más sencilla de generar un conjunto inventado es usar una distribución uniforme, generando los valores nominales y cualquier valor numérico dentro del rango con la misma probabilidad. Esta distribución es exhaustiva, pero nada fiel a los datos originales.
- Distribución normal: igual que la anterior, pero para los atributos numéricos se utilizará una distribución normal. Lógicamente, si en el caso anterior sólo era necesario conocer el máximo y el mínimo de cada atributo numérico, en este caso hará falta estimar una media y varianza a partir de los datos originales. Esta distribución es exhaustiva, pero poco fiel a los datos originales.
- Distribución original por remuestreo: dado un conjunto de entrenamiento, se trata de elegir aleatoriamente cada atributo a partir de los datos existentes en el anterior, dando la misma probabilidad a cada valor aparecido en el conjunto de entrenamiento. Con ello, se asegura que los valores de los atributos nominales que han aparecido más veces en el conjunto de entrenamiento sean más frecuentes en el inventado. Algo similar ocurrirá con los valores numéricos, aunque en este caso habrá zonas nunca cubiertas. Por tanto, esta distribución no es exhaustiva para los datos numéricos, pero bastante fiel a los datos originales.

Una opción interesante es utilizar la distribución normal para los numéricos y el remuestreo con los nominales. Todas las técnicas anteriores suponen que los atributos son independientes, aspecto que no suele cumplirse en la realidad, pero que facilita mucho la generación de los ejemplos. Existen técnicas más complejas que permiten generar un conjunto de datos inventados más ajustado a la población original (como p.ej. el algoritmo EM [Dempster y otros, 1977]), pero que requieren un número importante de datos para estimar bien la distribución original de los datos.

Una vez decidido el método de generación de datos inventados, es necesario decidir cuántos. Teóricamente, se ha demostrado [Blanco-Vega, 2007] que si el tamaño de los datos inventados usando una técnica exhaustiva es lo suficientemente grande, el modelo mimético puede conseguir un 100 % de fidelidad respecto al oráculo. Esto, de

alguna manera, permite confiar en el método mimético como una forma general para la ingeniería inversa, o, dicho de otra manera, para cambiar la representación de un oráculo en otra equivalente.

No obstante, existe una gran limitación, recurrente ya en la extracción de conocimiento. Si bien teóricamente podemos conseguir una fidelidad total, cuanto mayor es el conjunto inventado, mayor será el tamaño del modelo mimético, y, por tanto, peor la comprensibilidad del modelo. Por tanto es necesario ajustar los factores que influyen en el método, como por ejemplo, el nivel de poda (en el caso de que el algoritmo de salida sea un árbol de decisión), el tamaño de los datos inventados En [Blanco-Vega, 2007] se estudian experimentalmente estos y otros factores para encontrar un ajuste de los mismos con el objeto de obtener un compromiso entre comprensibilidad y precisión, y se establecen “procedimientos” para aplicar el método mimético de una manera óptima.

Finalmente, la técnica mimética no sólo sirve para capturar la semántica de un modelo incomprendible y convertirlo en un modelo en forma de reglas. En [Blanco-Vega, 2007] se amplían las fronteras de aplicación del método mimético a la contextualización de modelos. La contextualización o adaptación es necesaria en la vida de todo modelo y pretende alargar la vida útil del mismo. Los cambios de contexto se pueden presentar de diferentes maneras, como por ejemplo, la aparición de nuevos datos que anteriormente no existían o no eran importantes, el cambio de formato de los datos cuando aparecen o desaparecen atributos o valores de clase, la modificación de los costes de los errores, etc. De esta forma, cualquier técnica de aprendizaje computacional se puede utilizar como método general de revisión y de adaptación de modelos, independientemente de cómo el modelo fue generado y sin usar los datos originales o la distribución de los mismos. Para ello, bastará realizar un “reentrenamiento” del modelo usando la filosofía del método mimético.

En este punto se ha presentado una manera general de extraer reglas a partir de cualquier sistema existente entendido como una caja negra. Este método, que denominamos mimético puede verse como una forma general de extraer una explicación (en forma de árboles de decisión o sistemas de reglas) de cualquier modelo existente sin considerar ningún mecanismo interno del mismo (esto es, considerándolo una caja negra). La técnica mimética puede verse también como un “reaprendizaje”, donde se generan datos que, al etiquetarse por el modelo existente, sirven para aprender un “nuevo” modelo que imita o “mimetiza” el original, o, si es necesario, adapta el original a una nueva situación.

18.3 Técnicas de extracción de reglas sobre redes neuronales

Las redes neuronales, que se han estudiado en el capítulo 15, son una técnica de aprendizaje ampliamente utilizada en áreas tan diversas como el comercio, la industria o la medicina. Parte de su éxito se debe a que es un método robusto que puede trabajar con ruido en los datos (algo habitual en aplicaciones reales, como ya hemos comentado en capítulos anteriores) y a que es capaz de compactar la información o

conocimiento adquirido (el cual queda implícitamente almacenado en la red), siendo al mismo tiempo fácil y rápido el usarlo luego. Sin embargo, también existen algunos inconvenientes que pueden limitar la aplicabilidad de esta técnica. Nos referimos principalmente a su incapacidad de mostrar de forma comprensible cómo se han obtenido las salidas. En particular, este inconveniente dificulta, por ejemplo, su utilización en las llamadas aplicaciones de “seguridad crítica” [Andrews y otros, 1995] (como las centrales energéticas, compañías aéreas o aplicaciones médicas) en las que el usuario tiene que ser capaz de evaluar la salida de la red neuronal para cualquier condición de entrada posible. También las redes neuronales plantean problemas a la hora de utilizarlas para descubrir relaciones o dependencias en los datos, ya que en última instancia, éstas son incomprensibles (y por lo tanto poco útiles) expresándose como un vector de pesos.

La forma más directa de dotar de comprensibilidad a una red neuronal ya entrenada es convertirla en un conjunto de reglas proposicionales (es decir, al estilo de las que hemos visto en el capítulo 17), lo que comúnmente se conoce como *rule extraction*. Éste ha sido un tema de investigación muy activo durante la última década, dando lugar a diferentes propuestas y sistemas de extracción de reglas, algunos de los cuales revisaremos en esta sección. Muchos son los factores que se han usado para comparar entre sí los distintos métodos de extracción [Craven y Shavlik, 1999; Duch y otros, 2004]. Los más habituales son:

- La potencia expresiva de las reglas extraídas, es decir, el tipo de reglas.
- La calidad de las reglas extraídas, en términos de precisión, fidelidad a la red subyacente, comprensibilidad y consistencia de las reglas.
- Si el método de extracción es local (analiza los nodos de la red uno a uno) o global (se analiza la red como un todo).
- La complejidad algorítmica del método.
- La generalidad del método, en el sentido de si se puede aplicar a cualquier tipo de red o, por contra, si impone alguna restricción a la arquitectura de la red o a su entrenamiento.
- El tipo de variables soportadas: booleanas, discretas o continuas.
- La escalabilidad, es decir, su capacidad para trabajar con redes con un gran número de entradas, de nodos y conexiones.

Atendiendo al primer factor, podemos distinguir entre los métodos que producen reglas de la forma *si ... entonces ... si no ...*, es decir, que usan como lenguaje de representación la lógica simbólica convencional (lógica proposicional a dos valores, analizada en el capítulo 2), y los métodos que describen los conceptos usando lógica borrosa (véase el capítulo 7). En esta sección vamos a referirnos a los primeros, mientras que los segundos serán tratados en la sección 18.5.

El tercer factor, la globalidad del método, nos permite distinguir entre dos esquemas generales de extracción de reglas desde una red neuronal: la aproximación en

la que se extrae un conjunto de reglas *globales* que caracterizan las salidas (clases) directamente a partir de las entradas, y la aproximación en la que se extraen reglas *locales* descomponiendo la red neuronal multicapa en una colección de redes de una sola capa, tal que una combinación de las reglas que describen las redes individuales permite describir la red completa. Algunos autores a menudo llaman a la primera aproximación *pedagógica* (especialmente si la aproximación es de caja negra) y a la segunda *descomposicional*.

Muchos de los métodos propuestos plantean la extracción de reglas como un problema de búsqueda en el que hay que explorar el espacio de las reglas candidatas. Entonces, cada una es testeada sobre la red para ver su validez. En muchos casos, las reglas se construyen como conjunciones de las características de entrada, las cuales suelen ser booleanas. Esto significa que si tenemos n posibles características X_i , entonces hay 3^n posibles reglas ya que en los antecedentes de las reglas cada característica puede no estar presente, o bien aparecer como X_i (positivamente) o como $\neg X_i$ (negativamente). Cuando el espacio de búsqueda está limitado a incluir únicamente X_i o $\neg X_i$, pero sin poder excluir ambos, entonces el número de reglas en el espacio es de 2^n . En cualquier caso, se trata de espacios grandes que hacen computacionalmente inviable una búsqueda exhaustiva, por lo que es necesario utilizar alguna heurística que los restrinja. Asimismo, también hay que evitar visitar un mismo nodo varias veces. A continuación resumimos las propuestas más significativas de los métodos globales y locales.

18.3.1 Métodos globales

Uno de los primeros métodos pedagógicos o globales fue propuesto por Saito y Nakano. En [Saito y Nakano, 1988] los autores usan un proceso de búsqueda primero en amplitud para construir las reglas. Para limitar la explosión combinatoria proponen dos heurísticas. La primera limita el número de literales positivos y negativos en el antecedente de las reglas, así como la profundidad del proceso de búsqueda. La segunda, propone considerar como válidas únicamente aquellas reglas cuyos antecedentes involucran literales presentes en el conjunto de entrenamiento. Básicamente, se procede de la siguiente forma. Todas aquellas entradas que aparezcan como literales positivos en la regla reciben un valor de activación de 1, mientras que el resto de las entradas toman el valor 0. Estos valores se propagan por la red para clasificar la instancia. Luego se hace lo mismo para los literales negativos, pero esta vez activando las correspondientes entradas a 1. Al final, si en el primer caso la red clasifica la instancia como perteneciente a la clase de interés y en el segundo caso como no perteneciente entonces la regla es válida. Como vemos, esta primera técnica no usa prácticamente la topología y pesos de la red, y puede considerarse de caja negra, a diferencia de las que veremos más adelante.

Una variante de esta técnica es el método propuesto en [Gallant, 1993], el cual difiere del de Saito y Nakano en la forma en que las reglas se validan con respecto a la red neuronal. En este caso, las entradas no especificadas en la regla podrían tener cualquier valor de los permitidos, lo cual se indica asociándoles un intervalo de activación que se calcula a partir de los ejemplos cubiertos por la regla. Este intervalo

se propaga por la red, de tal forma que se puede determinar el rango de activación de las salidas a partir del cual el procedimiento determina si la regla es válida o no.

En [Thrun, 1995], se define el método VIA (*validity interval analysis*), una generalización de la técnica anterior para redes artificiales con propagación hacia atrás, en la que los intervalos de validez se calculan como si se tratara de un problema de programación lineal. Inicialmente se asignan intervalos de activación a las unidades de la red, las cuales constituyen un conjunto de restricciones sobre los valores de entrada y las activaciones de las salidas. Entonces, los intervalos son refinados propagándolos hacia adelante y hacia atrás y eliminando aquellos valores que son inconsistentes con la red. Otra de las características de este método es que puede tratar con características de entrada con valores continuos. Se comienza con los valores presentes en los datos de entrenamiento y se van generalizando, reemplazándolos por intervalos cada vez más amplios, hasta alcanzar una buena generalización.

Algoritmo 18.1 Algoritmo REAL

```

Procedimiento REAL
para cada clase  $c$  hacer
     $R_c = 0;$ 
    fin para
    repetir
         $e = \text{ejemplos};$ 
         $c = \text{clasifica}(e);$ 
        si  $e$  no está cubierto por  $R_c$  entonces
             $r = \text{regla formada a partir de } e;$ 
            para cada antecedente  $r_i$  de  $r$  hacer
                 $r' = r \sim r_i; \{\text{eliminar } r_i \text{ de } r\}$ 
                si  $\text{subconjunto}(c, r') = \text{true}$  entonces
                     $r = r';$ 
                fin si
            fin para
             $R_c = R_c \cup r;$ 
        fin si
    hasta criterio de parada

```

Otra aproximación alternativa es la propuesta por Craven y Shavlik [Craven y Shavlik, 1994] bajo el nombre “extracción de reglas como aprendizaje” (*rule-extraction-as-learning*, REAL), la cual también se puede considerar de caja negra. La idea básica es ver la extracción de reglas como una tarea de aprendizaje cuya función objetivo es la función computada por la red, y cuyas entradas son las entradas de la red. Cuando un ejemplo no es cubierto por ninguna regla entonces se crea una nueva regla y luego se testea en la red. Para ello, el procedimiento que determina si una regla es válida toma como entrada una regla r y una etiqueta de clase c y devuelve “cierto” si todas las instancias cubiertas por r son de clase c . Asimismo, se proponen dos criterios para parar la generación de las reglas: cuando las reglas constituyen un modelo de la red neuronal lo suficientemente preciso, o bien cuando después de varias iteraciones no se han producido reglas nuevas. El Algoritmo 18.1 resume esquemáticamente este método.

En el Algoritmo 18.1 la función “ejemplos” genera ejemplos de entrenamiento para la extracción de las reglas (los cuales pueden ser ejemplos pertenecientes al conjunto usado para entrenar la red, o bien una muestra aleatoria, o bien ejemplos generados aleatoriamente pero de una clase específica), mientras que “subconjunto(\cdot, \cdot)” es la función que comprueba la validez de la regla creada.

Algoritmo 18.2 Algoritmo RULENEG

Procedimiento RULENEG

$R = \emptyset;$

para cada patrón s del conjunto de entrenamiento **hacer**
 $c = \text{clasifica-red}(s); \{ \text{ se clasifica } s \text{ usando la red neuronal} \}$
si s no es clasificado por ninguna regla de R **entonces**
 inicializar una nueva regla r para c ;
para cada entrada i en la red **hacer**
 $s' = s;$
 negar la i -ésima entrada en s' ;
 $c' = \text{clasifica-red}(s');$
 si c es diferente de c' **entonces**
 añadir a r la entrada i y su valor;
 fin si
 fin para
fin si
 $R = R \cup r$
fin para

Otro ejemplo de método pedagógico, basado en el mismo principio que REAL, es RULENEG [Pop y otros, 1994]. En este caso, por cada ejemplo no clasificado por las reglas existentes, se crea una regla cuyo antecedente es la conjunción de todos aquellos literales de entrada que influyen en la clase. Para determinar qué literales son los influyentes se van negando uno a uno los valores de entrada comprobándose si la clase cambia. En el Algoritmo 18.2 se esboza esquemáticamente este procedimiento.

Veamos un ejemplo que ilustra cómo funciona la técnica RULENEG. Supongamos el siguiente conjunto de entrenamiento de un problema binario (clases + y -):

Instancia	Estatura	Pelo	Ojos	Clase
e1	baja	rubio	azules	+
e2	alta	rubio	castaños	-
e3	alta	pelirrojo	azules	+
e4	baja	moreno	azules	-
e5	alta	moreno	azules	-
e6	alta	rubio	azules	+
e7	alta	moreno	castaños	-
e8	baja	rubio	castaños	-

Tabla 18.1: Datos de entrenamiento de un problema binario.

Con estos datos entrenamos una red neuronal cuya funcionalidad podemos expresar con las siguientes reglas:

Regla 1: *si* pelo=moreno *entonces* clase=-

Regla 2: *si* pelo=pelirrojo *entonces* clase=+

Regla 3: *si pelo=rubio y ojos=azules entonces clase=+*

Regla 3: *si pelo=rubio y ojos=marrones entonces clase=-*

Comenzamos con R vacío. Tomamos e_1 y lo clasificamos con la red obteniendo $c = +$. Como e_1 no está cubierto por R hacemos $s' = e_1$. A continuación analizamos los atributos de s' :

- Negamos el primer atributo en s' : (no baja, rubio, azules), y clasificamos este nuevo ejemplo con la red dándonos una clase $c' = +$; como c es igual a c' este atributo (la estatura) no es relevante.
- Negamos el segundo atributo en s' : (baja, no rubio, azules), y clasificamos este nuevo ejemplo con la red dándonos una clase $c' = -$; como ha cambiado la clase la condición (pelo=rubio) debe aparecer en la regla.
- Negamos el tercer atributo en s' : (baja, rubio, no azules), y clasificamos este nuevo ejemplo con la red dándonos una clase $c' = -$; como ha cambiado la clase la condición (ojos=azules) debe aparecer en la regla.

Por lo tanto, la regla generada es

si pelo=rubio y ojos=azules entonces clase=+

que coincide con la Regla 3 que describía la red.

La técnica DEDEC [Tickle y otros, 1996] extrae reglas encontrando la mínima información necesaria para distinguir un objeto particular de los otros objetos. Para ello se crea un nuevo conjunto de entrenamiento clasificado por la red neuronal. Estos ejemplos se ordenan por la importancia de sus atributos, la cual se establece analizando los vectores de peso de las entradas de la red neuronal y su influencia en las salidas. A continuación, se crean grupos de ejemplos con atributos importantes usando un algoritmo de agrupamiento (como, por ejemplo, el algoritmo k-NN) y se usan solamente estos atributos para crear las reglas. Como criterios de parada esta técnica usa uno basado en la estabilidad (parar cuando ya no se mejora) y otro basado en la importancia de los ejemplos (por debajo de un cierto umbral ya no se sigue procesando ejemplos).

Otro de los métodos pedagógicos que revisamos en esta sección está muy relacionado con los métodos de caja negra vistos en la sección 18.2, y en particular con la técnica mimética. Se trata de TREPAN [Craven y Shavlik, 1996], un sistema que considera la red neuronal como un oráculo y que usa la inducción de árboles de decisión como método de extracción de reglas. Básicamente, la idea es inventar un conjunto de datos que son clasificados por la red neuronal. Estos datos etiquetados junto con el conjunto de entrenamiento inicial de la red neuronal sirven para entrenar un árbol de decisión. Las principales diferencias entre TREPAN y los algoritmos tradicionales de inducción de árboles de decisión (véase el capítulo 17) son:

- En TREPAN el árbol es construido siguiendo una estrategia “primero el mejor”. Para seleccionar el nodo “mejor” se usa un criterio basado en incrementar la

fidelidad del árbol generado a la red neuronal. En concreto, la función que se utiliza para evaluar un nodo N es

$$f(N) = \text{reach}(N) \times (1 - \text{fidelity}(N))$$

donde $\text{reach}(N)$ es la fracción estimada de instancias que alcanzan el nodo N cuando pasan a través del árbol, y $\text{fidelity}(N)$ es el grado de concordancia en la clasificación de esas instancias por el árbol y por la red.

- En TREPAN los nodos del árbol son expandidos sólo cuando un número considerable de vectores (ejemplos) que caen en ese nodo han sido analizados (del orden de 10^3).
- En TREPAN se usan tres criterios de parada: un test estadístico para determinar con una alta probabilidad que un nodo cubre sólo instancias de una clase, en cuyo caso ya no se expande el nodo; un parámetro (en términos del número de nodos internos) que fija el usuario y que limita la talla del árbol; finalmente, ya que TREPAN expande el árbol explorando el nodo mejor, cada árbol producido difiere del anterior únicamente en el subárbol correspondiente al nodo expandido en último lugar, hecho que usa TREPAN para devolver el árbol de la secuencia que es más fiel a la red con respecto a un conjunto de validación usado a tal efecto.

Existen más métodos que convierten una red neuronal en un árbol de decisión, como por ejemplo DecText (*Decision Tree Extractor*) [Boz, 2002]. Por otro lado, el algoritmo BRAINNE [Sestito y Dillon, 1994] es otra muestra de la aproximación pedagógica que se aplica a redes neuronales con propagación hacia atrás. La idea es generar las reglas atendiendo a la proximidad entre las entradas y las salidas de la red. En concreto, extrae una regla para cada salida determinando las entradas o atributos que contribuyen a esa salida, las cuales constituyen el antecedente de la regla. Otra característica de este algoritmo es que trabaja directamente con valores continuos sin necesidad de discretizarlos previamente. El propio método segmenta los atributos continuos en discretos y extrae las reglas directamente.

18.3.2 Métodos locales

Los métodos locales o descomposicionales analizan fragmentos de la red (generalmente los nodos ocultos y las salidas de forma individual) para extraer reglas. Estas redes se basan generalmente en funciones sigmoidales o en funciones localizadas. En cualquier caso, la salida de cada neurona debe poder expresarse como un valor binario (sí/no) que constituye el consecuente de la regla. Así, cada unidad de salida u oculta se interpreta como una regla booleana, lo que reduce el problema de extraer las reglas a un problema de determinar las situaciones bajo las cuales la regla es cierta. Nótese que, como generalmente los valores de las funciones de activación son o bien próximos a 0 o bien próximos a 1, entonces la búsqueda se reduce a analizar los vectores de peso. Algunas de las primeras implementaciones de esta aproximación fueron el algoritmo KT [Fu, 1991] y el *subset algorithm* propuesto por Towell y Shavlik [Towell y Shavlik,

1993]. El nombre de este último se debe a que busca explícitamente subconjuntos de entradas cuyos pesos excedan los umbrales de activación de las unidades. Para ello, comienza considerando conjuntos conteniendo un simple enlace si con él se garantiza que se excede el umbral, en cuyo caso se reescribe como regla. La búsqueda prosigue considerando luego subconjuntos de dos elementos y así sucesivamente hasta que se hayan explorado todos los subconjuntos posibles. Finalmente, se eliminan las reglas subsumidas por otras. El Algoritmo 18.3 muestra el esquema básico de funcionamiento del algoritmo subconjunto.

Algoritmo 18.3 Algoritmo subconjunto

```

Procedimiento SUBSET
para cada unidad oculta y de salida hacer
     $S_P$ =todos los subconjuntos de atributos de entrada con pesos positivos cuya suma excede el
    umbral de activación de la unidad;
    para cada  $p \in S_P$  hacer
        buscar un conjunto de atributos negativos  $S_N$  tales que la diferencia entre la suma positiva
        de  $p$  y  $N - n$  (donde  $N$  es la suma de los atributos negativos y  $n$  es un elemento de  $S_N$ ) supera el umbral de la unidad;
        para cada  $n \in S_N$  hacer
            construir la regla "si  $p$  y no  $n$  entonces el concepto designado por la unidad"
        fin para
    fin para
fin para

```

El inconveniente de esta forma de proceder es que el algoritmo es exponencial (para cada nodo, el espacio de búsqueda está limitado a reglas que incluyan o bien x_i o bien $\neg x_i$, para cada enlace x_i , es decir 2^n posibles reglas si hay n enlaces, tal y como se ha comentado anteriormente). Fu resolvió parcialmente este problema limitando el número de antecedentes de las reglas generadas. En cambio, la solución adoptada por Towell y Shavlik fue el algoritmo *M-of-N*, llamado así porque las reglas son de la forma: *si* (M de los siguientes N antecedentes son ciertos) *entonces* ...

El siguiente ejemplo de [Darbari, 2001] muestra la reducción de reglas que se puede obtener usando reglas del tipo *M-of-N*. Supongamos un nodo *A* con umbral=5 y 4 enlaces (*B*, *C*, *D* y *E*) con los pesos indicados en la parte izquierda de la Figura 18.3. Las reglas obtenidas por el algoritmo *subset* se muestran a la derecha de la Figura 18.3. Estas reglas podrían reemplazarse por:

si 3 de {B, C, D, no(E)} entonces A

que expresa mucho más claramente las condiciones sobre *A*. Para ello, el algoritmo *M-of-N* procede de la siguiente forma:

1. Para cada nodo interno y salida se hacen grupos de entradas con pesos similares.
2. Se calcula el peso medio de cada grupo y se asigna como peso a todos los miembros del grupo.
3. Se eliminan los grupos que no afectan a la activación o no del nodo o salida.

4. Dejando los pesos de los enlaces constantes, se optimizan los umbrales de los nodos y salidas usando el algoritmo de propagación hacia atrás (esto producirá una red con menos entradas independientes y, por consiguiente, más fácil de analizar).
5. Se crea una regla por cada nodo oculto y salida, la cual constará de un umbral en función de los umbrales y pesos de los enlaces que hayan quedado.
6. Se simplifican las reglas cuando sea posible.

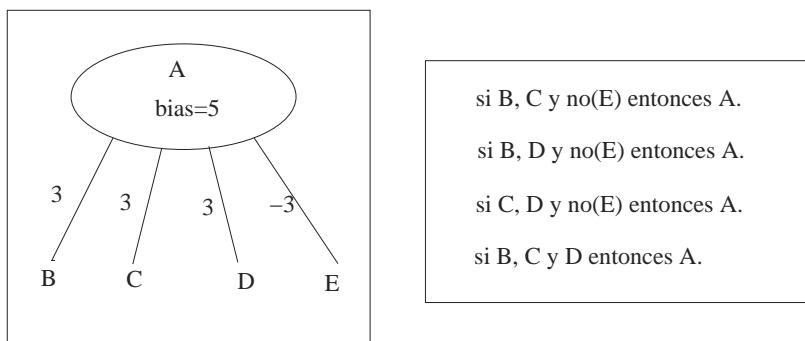


Figura 18.3: Extracción de reglas con el algoritmo *subset*.

Ambos algoritmos (*subset* y *M-of-N*) están incorporados al sistema KBANN (*Knowledge Based Artificial Neural Network*) [Shavlik y Towell, 1994]. Este sistema se considera un marco de aprendizaje híbrido (en el sentido que mezcla reglas construidas a mano y ejemplos clasificados durante el aprendizaje) en el que primero se entrena la red a partir de las especificaciones y reglas que se le suministran y luego se refina con los dos algoritmos mencionados.

Para finalizar esta sección mencionaremos algunas técnicas aplicables a determinados tipos de redes neuronales. Por ejemplo, la técnica RuleNet [McMillan y otros, 1991] consiste en efectuar los siguientes pasos de forma iterativa: a) entrenar la red neuronal, b) extraer reglas simbólicas, c) propagar hacia atrás las reglas obtenidas en la red. Este proceso termina cuando las reglas extraídas caracterizan adecuadamente el dominio del problema. Este método se aplica únicamente a redes cuya arquitectura consta de una capa de entrada, una de salida y una intermedia, por lo que no resulta general. Otra alternativa es la técnica Rulex [Andrews y Geva, 1995a,b], la cual fue diseñada para explotar la forma de construir y el comportamiento posterior de cierto tipo particular de perceptrón multicapa (MLP), las redes *constrained error back-propagation* (CEBP) que clasifican de una forma similar a como lo hacen las redes de función de base radial. Las unidades ocultas de las CEBP son unidades de respuesta localmente basadas en funciones sigmoidales (LRU), lo que hace que el conjunto de entrenamiento quede partido en regiones disjuntas, cada región representada

por una simple unidad oculta. Las reglas se extraen directamente de los pesos y umbralas de activación. Otros ejemplos son el algoritmo *MLP2LN* [Duch y otros, 1998] para extraer reglas desde redes MLP, y las propuestas para tratar redes neuronales recurrentes (véase [Jacobsson, 2005] para un resumen de las más significativas).

18.4 Técnicas de extracción de reglas sobre otros paradigmas

Aunque la mayoría de las técnicas propuestas en la última década se refieren a métodos para extraer reglas proposicionales desde redes neuronales, también se han hecho algunas propuestas para convertir otro tipo de modelos en un conjunto de reglas. Uno de estos modelos es el producido por las máquinas de vectores soporte (*Support Vector Machines, SVM*), analizadas en el capítulo 15. Esta técnica se ha vuelto muy popular, ya que permite abordar problemas cuyos datos tienen una alta dimensionalidad.

Como hemos visto en la sección anterior, las técnicas de extracción de reglas a partir de redes neuronales se agrupan en pedagógicas (o basadas en el aprendizaje) y descomposicionales. Esta distinción sigue siendo válida para las SVM, en las que las aproximaciones descomposicionales se basan en el análisis de los vectores soporte generados por la SVM, mientras que las aproximaciones globales aprenden lo que ha aprendido la SVM.

En [Barakat y Diederich, 2004] se presenta un método global similar a la técnica TREPAN vista para las redes neuronales. Básicamente, la idea es el método mimético presentado en la sección 18.2 y consiste en:

- Entrenar la SVM con unos datos de entrada etiquetados.
- Generar más datos (aleatoriamente) con los mismos atributos pero diferentes valores que son clasificados por la SVM.
- Con ambos datasets se entrena un nuevo modelo usando cualquier técnica que sea comprensible (como un árbol de decisión); en concreto, los autores usan el algoritmo C5 para los experimentos sobre un dataset médico.

Un ejemplo de técnica descomposicional es la descrita en [Núñez y otros, 2002]. En este trabajo se propone combinar la SVM con un método de obtención de prototipos. Más exactamente, una vez generada la SVM, se aplica un algoritmo de agrupamiento de los vectores soporte obtenidos para determinar vectores prototipos para cada clase. Combinando los vectores soporte con los prototipos mediante métodos geométricos se consigue partir el espacio de entrada en regiones elipsoides o hiperrectangulares. Cada una de estas regiones define una regla. Dicha regla es la ecuación de una elipse (si la región es elipsoidal), como por ejemplo “si $AX_1^2 + BX_2^2 + CX_1X_2 + DX_1 + EX_2 + F$ entonces *clase*” siendo el espacio bidimensional (con dos entradas X_1 y X_2), o bien es una regla con intervalos si la región es un hiperrectángulo, como por ejemplo “si $X_1 \in [a..b]$ y $X_2 \in [c..d]$ entonces *clase*”. Para determinar el número de elipsoides

por clase se comienza con un prototipo y se calcula la región. Se aplica un test de partición a la misma, si es negativo se extrae la regla y en caso contrario se parte la región. El proceso continúa hasta que las regiones se ajustan al criterio de partición o bien hasta que se alcanza el número máximo de iteraciones. Para partir las regiones cuyo test de partición ha resultado positivo se vuelven a calcular prototipos a partir de los datos de esas regiones y se procede como hemos indicado antes.

Otra aproximación aplicable no sólo a las SVM lineales sino a cualquier clasificador lineal (como, por ejemplo, los discriminantes lineales de Fisher [Mika y otros, 1999]) es el introducido en [Fung y otros, 2005]. En este caso se obtienen reglas que no solapan y que describen regiones delimitadas por hiperplanos, por lo que las reglas son del tipo intervalo, como hemos visto antes. Estas reglas se extraen resolviendo problemas de programación lineal con $2n$ variables, donde n es el número de características de entrada. Cada iteración del algoritmo de extracción se formula como uno de los dos problemas de optimización posibles, correspondientes a estos dos criterios de optimallidad sobre las reglas: que se maximice el volumen cubierto por una regla (esto es, un problema de optimización no lineal con restricciones), y que se maximice el número de ejemplos cubiertos por cada regla (esto es, un problema de programación lineal). Finalmente, para conseguir que las reglas sean disjuntas se exige que cada hipercubo tenga un punto en el hiperplano de separación.

En [Barakat y Diederich, 2005] se presenta un método híbrido, que incorpora elementos de las aproximaciones descomposicionales y pedagógicas. La tarea de extracción de las reglas se hace en tres pasos:

1. Entrenar la SVM con los datos de entrenamiento.
2. Crear las reglas a partir de los vectores soporte y los parámetros asociados:
 - Del conjunto de datos de entrenamiento seleccionar los patrones convertidos en vectores soporte eliminando la clase.
 - Reclasificar los datos del paso anterior usando la SVM.
 - Usar este conjunto de datos para entrenar un modelo que pueda expresarse mediante reglas simbólicas (como el algoritmo de inducción de árboles de decisión C5).
3. Evaluar la calidad de las reglas extraídas.

La técnica ha sido experimentada con varios datasets médicos del repositorio UCI [Newman y otros, 1998] obteniéndose reglas con buena precisión que, en algunos casos, hacen mejores generalizaciones que la propia SVM de la que se extrajeron.

18.5 Técnicas de extracción de reglas borrosas

En el capítulo 7 de este libro vimos cómo las reglas borrosas permiten representar conocimiento aproximado o impreciso de una manera más adecuada que las reglas no borrosas. Esta ventaja es claramente aprovechable para el problema de adquisición

de conocimiento a partir de expertos, ya que éstos suelen tener un conocimiento aproximado o impreciso sobre el cual basan sus decisiones. Lo mismo se puede decir de los casos en los que queremos extraer reglas a partir de modelos complejos, como por ejemplo las redes neuronales. Generalmente, las reglas simples (no borrosas) no son capaces de capturar fielmente el modelo, a no ser que utilicen un número muy alto de reglas. Aun así, sufren el problema del sobreajuste, al tratar de fijar de manera precisa aquello que no lo es en origen.

Por tanto, las técnicas de extracción de sistemas de reglas borrosas son muy apropiadas para las aplicaciones que hemos comentado en este capítulo. Como existen numerosas técnicas de extracción de este tipo de reglas, nos centraremos en analizar un ejemplo representativo de métodos de extracción de sistemas de reglas independientemente del modelo original (como un problema de aprendizaje, a partir de ejemplos), que denominaremos de “caja negra” y un ejemplo de técnica de las denominadas “neuro-fuzzy”, que permiten extraer reglas borrosas de un tipo específico de sistema de origen, en particular una red neuronal.

18.5.1 Técnicas de extracción directa (caja negra)

Una de las múltiples aproximaciones para obtener un conjunto de reglas borrosas a partir de un conjunto de datos es la que se basa en computación evolutiva y que ha dado lugar a la aproximación frecuentemente denominada sistemas genéticos borrosos o GFS, del inglés “genetic fuzzy systems”. Un GFS es un sistema de aprendizaje basado en un algoritmo genético que obtiene o refina un conjunto de reglas borrosas. Los algoritmos genéticos, como vimos en el capítulo 11, son algoritmos inspirados en la genética, que permiten adaptarse y obtener soluciones en problemas complejos y, especialmente, donde se tiene que optimizar un conjunto o una combinación de indicadores. Existen muchas variantes de algoritmos genéticos, dependiendo de los operadores de cruce y de mutación, del criterio de selección, del tamaño de las poblaciones y, muy especialmente, de la codificación.

Siguiendo la clasificación de [Cordón y otros, 2004, 2001] existen dos tipos principales de algoritmos genéticos para el aprendizaje de sistemas de reglas borrosas, atendiendo a si sólo aprenden la base de reglas o si también aprenden los factores de escalado y las funciones de pertenencia de los conjuntos borrosos asociados con las etiquetas lingüísticas.

Respecto a los primeros, los que aprenden genéticamente la base de reglas, se ha de asumir un conjunto de funciones de pertenencia en la base de datos a la que las reglas se referirán por medio de etiquetas lingüísticas. Las tres aproximaciones habituales están presentes en estos sistemas: *Michigan* [Bonarini, 1996; Ishibuchi y otros, 1999; Valenzuela-Rendón, 1991], *Pittsburgh* [Hoffmann y Pfister, 1997; Pham y Karaboga, 1991; Thrift, 1991] y la aproximación de aprendizaje iterativo de reglas [Cordón y Herrera, 1997; González y Pérez, 1999]. La base de reglas se puede representar como una matriz relacional [Thrift, 1991], una tabla de decisión [Pham y Karaboga, 1991], siendo estas dos aplicables en la aproximación Pittsburgh únicamente, o, simplemente como una lista de reglas [González y Pérez, 1999; Hoffmann y Pfister, 1997; Valenzuela-Rendón, 1991], que es la representación más extendida. Cada regla in-

dividual puede entonces codificarse en forma disyuntiva normal (DNF) representada como una cadena binaria de longitud fija [González y Pérez, 1999; Valenzuela-Rendón, 1991] o con códigos de longitud variable [Hoffmann y Pfister, 1997].

Respecto a los segundos, los que aprenden no sólo la base de reglas, sino también los factores de escalado y las funciones de pertenencia, se han utilizado representaciones con cromosomas de longitud variable, genomas de multicromosomas y cromosomas que codifican las reglas una a una. Se han utilizado también los tres tipos más importantes de algoritmo genético: aproximación *Michigan* [Parodi y Bonelli, 1993; Velasco, 1998], aproximación *Pittsburgh* [Carse y otros, 1996; Lee y Takagi, 1993; Magdalena y Monasterio-Huelin, 1997; Park y otros, 1994], y la aproximación de aprendizaje iterativo de reglas [Cordón y otros, 1998; Cordón y Herrera, 1997; Cordón y Herrera, 1999]. También existen diferentes variantes a la hora de representar las reglas. Por ejemplo, muchos sistemas, como [Carse y otros, 1996; Cordón y Herrera, 1997; Magdalena y Monasterio-Huelin, 1997; Park y otros, 1994; Velasco, 1998], son de tipo Mamdani, donde existe una base de datos con los factores de escalado y las funciones de pertenencia de los conjuntos borrosos asociados con las etiquetas lingüísticas, aparte de la base de reglas. Otros son de los que se denominan TSK (de Takagi-Sugeno-Kang) [Cordón y Herrera, 1999; Lee y Takagi, 1993], donde la salida es función de las variables de entrada (al estilo de una regresión lineal) y no una variable lingüística.

Pasemos a ver algún ejemplo del tipo Mamdani. En este caso, las reglas del conjunto de reglas extraído tienen la forma:

$$R_i : \text{SI } x_1 \text{ ES } A_{i1} \wedge x_2 \text{ ES } A_{i2} \wedge \dots \wedge x_n \text{ ES } A_{in} \text{ ENTONCES } y \text{ ES } B_i$$

donde $x_1 \dots x_n$ son las variables de entrada, y es la variable de salida, y las A_{ij} y las B_i son variables lingüísticas que corresponden (borrosamente) con los valores de entrada y de salida respectivamente.

Un ejemplo de regla de tipo Mamdani en una cámara frigorífica podría ser la siguiente:

$$\begin{aligned} R_1 : & \text{ SI } \text{presión ES "bajo"} \wedge \\ & \text{temperatura ES "muy bajo"} \wedge \\ & \text{concentración_dióxido ES "muy alto"} \\ & \text{ENTONCES duración ES "alto"} \end{aligned}$$

donde cada atributo de entrada (*presión*, *temperatura*, *concentración_dióxido*) y de salida (*duración*) tiene definido unos rangos para cada etiqueta lingüística. Por ejemplo, los conjuntos borrosos para la temperatura podrían ser los que se muestran en la descomposición triangular de la Figura 18.4.

Dependiendo de la manera en la que pueden definirse los términos lingüísticos tenemos dos tipos de sistemas de reglas borrosas.

- La primera aproximación, denominada *descriptiva*, considera que las A_{ij} tienen el mismo conjunto borroso para todas las reglas y que, generalmente, coinciden con una representación lingüística existente ya en el contexto del problema.

- La segunda aproximación, denominada reglas de tipo Mamdani aproximadas, permiten que cada regla defina sus conjuntos borrosos para las A_{ij} y las B_i .

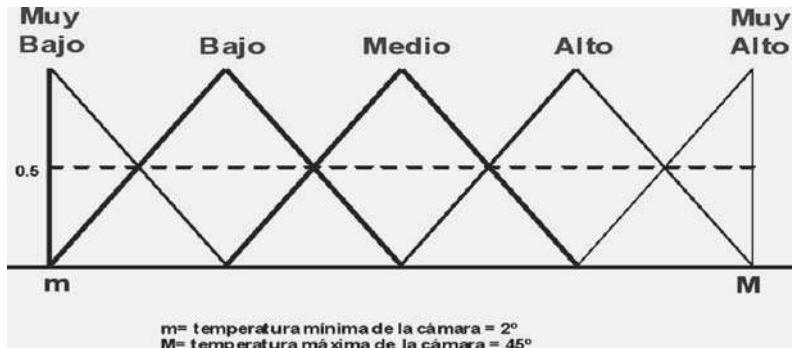


Figura 18.4: Descomposición triangular del atributo temperatura.

Lógicamente, la primera aproximación facilita la comprensión del conjunto de reglas obtenido, aunque tiene limitaciones para problemas complejos y de alta dimensionalidad. La segunda aproximación permite atacar con mayores expectativas estos tipos de problemas, pero normalmente a costa de un conjunto de reglas más difícil de entender, al utilizar cada regla diferentes términos lingüísticos. Por ejemplo, para el caso anterior, podríamos tener dos reglas que usaran el atributo de la temperatura en las condiciones de la regla y que tuvieran etiquetas diferentes o, peor aún, límites diferentes para las etiquetas “Muy bajo”, “bajo”, “medio”, “alto”, “muy alto”.

Finalmente, los algoritmos genéticos no son la única técnica que se ha utilizado para extraer reglas borrosas. Existen muchas otras técnicas, no sólo evolutivas (como por ejemplo, las de programación genética [ter Bor, 2001; Tunstel y Jamshidi, 1996]), sino basadas en árboles de decisión u otro tipo de paradigmas clásicos de aprendizaje de reglas extendidos a reglas borrosas [Cordón y otros, 2001].

Por ejemplo, el sistema “genetic programming extractor” de [ter Bor, 2001], extrae las reglas de la Figura 18.5 a partir del conjunto de datos “iris” del repositorio UCI, donde CF representa la confianza otorgada por el método a la regla.

Iris plants IF x3 IS small THEN Class1 WITH CF 0,58 IF x3 IS medium THEN Class2 WITH CF 0,81 IF x3 IS medium_large THEN Class3 WITH CF 0,31 IF x4 IS medium_large THEN Class3 WITH CF 0,81
--

Figura 18.5: Conjunto de reglas extraídas por el sistema “genetic programming extractor” para el problema de las plantas *iris*.

18.5.2 Sistemas neuroborrosos (neuro-fuzzy)

Otro de los paradigmas de extracción de reglas borrosas que, por su importancia, requiere una mención especial es lo que se denomina conjuntamente “técnicas neuroborrosas” (del inglés, neuro-fuzzy) [Jang y otros, 1997; Kosko, 1992; Lin y Lee, 1996; Nauck y otros, 1997]. Aunque el término se utiliza para cualquier híbrido entre las técnicas de redes neuronales artificiales y las técnicas de conjuntos o reglas borrosas (incluyendo técnicas borrosas para la optimización de redes neuronales, modificación de las categorías y funciones de pertenencias borrosas usando redes neuronales, etc.), en este apartado nos centraremos exclusivamente en la extracción de reglas borrosas a partir de redes neuronales. En este sentido, los sistemas que integran las redes neuronales pero proporcionan una representación basada en reglas borrosas se denominan sistemas neuroborrosos.

Un sistema neuroborroso incorpora el razonamiento de los sistemas borrosos mediante conjuntos borrosos y un modelo lingüístico basado en reglas SI-ENTONCES. La potencia de esta aproximación es que tiene un modelo interno universal (como el que proporcionan las redes neuronales) pero con la capacidad de representarse en forma de reglas. Lógicamente, la fidelidad del conjunto de reglas borrosas extraído suele ir en detrimento de la comprensibilidad de las mismas. Es decir, si queremos un modelo borroso fiel a la red neuronal original obtendremos generalmente muchas reglas y categorías. Si queremos un modelo borroso más simple, habremos de sacrificar algo de fidelidad.

Al igual que las técnicas de extracción de reglas (crisp) a partir de redes neuronales vistas en la sección 18.3, también existen numerosísimas aproximaciones neuroborrosas para extraer reglas borrosas a partir de una red neuronal. En unas aproximaciones, una red neuronal multicapa con función de activación sigmoidal típica puede ser utilizada como base de la extracción de reglas. En otras aproximaciones, es necesario que la red neuronal cumpla algún tipo de requisitos especiales en su modo de entrenamiento o topología, o, por el contrario, que el tipo de reglas borrosas sean de alguna familia especial.

Respecto a las primeras podemos citar el método de Ishibuchi [Ishibuchi y otros, 1995, 1999], como uno de los métodos clásicos de extracción de reglas a partir de redes neuronales. La red de partida puede ser cualquier perceptrón multicapa, independientemente del modo de entrenamiento o de topología (siempre que no sea recursiva). El método se basa en que debe ya existir una descomposición que defina los conjuntos borrosos para cada atributo. El procedimiento original utiliza una aproximación de fuerza bruta en la que se evalúan todas las combinaciones de vectores de entradas lingüísticas según los conjuntos borrosos. Si una combinación lingüística es válida, la regla correspondiente se añade al conjunto de reglas. Para determinar si una combinación lingüística es válida se tienen que proyectar las entradas de cada combinación con las salidas, usando lo que se denominan cortes α [Ishibuchi y otros, 1995] y después se extiende también la regla de decisión usando un umbral. Por ejemplo, en la Figura 18.6, se ve una red neuronal con dos entradas (x_1 y x_2) con conjuntos borrosos triangulares de etiquetas “pequeño”, “mediano” y “grande”, dos unidades ocultas y tres salidas (y_1 , y_2 , y_3), en las que se está probando la combinación “grande”, “mediano”.

A partir del umbral una posible regla que se podría extraer es:

$$R_1 : \text{SI } x_1 \text{ ES "grande"} \wedge x_2 \text{ ES "mediano"} \text{ ENTONCES clase} = y_2$$

Lógicamente, la aproximación original de Ishibuchi es simple pero tiene un coste computacional prohibitivo cuando el número de variables de entrada y de categorías borrosas por atributo aumenta. Diversos trabajos posteriores, incluyendo trabajos del propio Ishibuchi [Ishibuchi y otros, 1999], proponen diversas heurísticas para no probar todas las combinaciones.

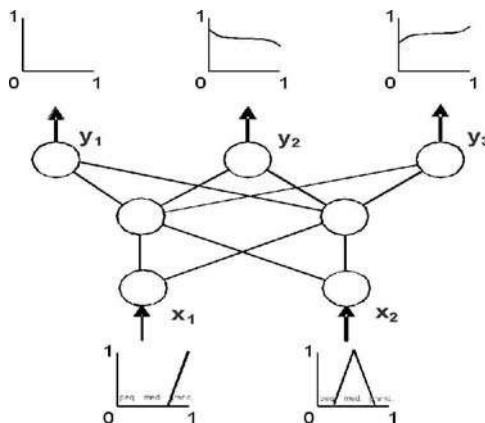


Figura 18.6: Proyección borrosa a través de una red neuronal.

Por ejemplo, en la Figura 18.7 se muestra el conjunto de reglas que extrae el método de Ishibuchi para el problema de la planta Iris. En dicho conjunto, CF representa la confianza otorgada por el método a la regla.

```
Iris plants
IF x3 IS large THEN Class3 WITH CF 0,98
IF x4 IS large THEN Class3 WITH CF 0,90
IF x3 IS small AND x4 IS small THEN Class1 WITH CF 0,03
IF x3 IS medium AND x4 IS medium THEN Class2 WITH CF 0,31
IF x2 IS small AND x3 IS medium-large THEN Class3 WITH CF 0,40
IF x2 IS small AND x4 IS medium-large THEN Class3 WITH CF 0,65
IF x2 IS medium-small AND x4 IS medium-large THEN Class3 WITH CF 0,39
IF x3 IS medium-large AND x4 IS small THEN Class3 WITH CF 0,74
IF x3 IS medium-large AND x4 IS medium-small THEN Class3 WITH CF 0,49
```

Figura 18.7: Conjunto de reglas extraídas por el método de Ishibuchi para el problema de las plantas *iris*.

Finalmente, respecto a las aproximaciones que exijan que la red neuronal cumpla algún tipo de requisitos especiales en su modo de entrenamiento o topología, o, por el contrario, que el tipo de reglas borrosas sean de alguna familia especial, es interesante nombrar los trabajos que intentan establecer la equivalencia de algunos tipos de redes neuronales con algunos tipos de sistemas de reglas borrosas. Por ejemplo, Jang y Sun [Jan y Sun, 1993] usan funciones de pertenencia gausianas para mostrar que una red de función de base radial es equivalente a un sistema borroso. [Benítez y otros, 1997] muestran la equivalencia entre una red neuronal con activación con función logística con una base de reglas borrosas de tipo Mamdani. Trabajos posteriores han generalizado los resultados [Castro y otros, 2002; Kolman y Margaliot, 2005]. No obstante, no siempre se asegura una fácil interpretabilidad del conjunto de reglas extraídos ni un tamaño de las mismas que sea siempre manejable.

18.6 Lecturas recomendadas

Para comprender y, fundamentalmente, complementar lo tratado en este capítulo se recomiendan dos libros de base. El primero de ellos, el libro de Greenwell [Greenwell 1988] es un clásico sobre la ingeniería del conocimiento, que pone de relieve la necesidad de gestionar el conocimiento, los problemas del cuello de botella en la adquisición y que una posible solución son las técnicas inductivas del aprendizaje automático, la estadística o la minería de datos. Respecto a esta última, el libro *Introducción a la Minería de Datos* [Hernández y otros, 2004] cubre las técnicas más importantes en este ámbito.

Respecto a aspectos más concretos de este capítulo, se puede profundizar en la técnica mimética y, en general, en la extracción por técnicas genéricas de caja negra a partir de [Blanco-Vega, 2007]. En cuanto a las técnicas específicas para redes neuronales, se pueden consultar los siguientes “surveys”: [Andrews y otros, 1995; Darbari, 2001; Duch y otros, 2004; Mitra y otros, 2002]. La extracción de reglas borrosas usando técnicas genéticas tiene una fuente muy apropiada en [Cordón y otros, 2001].

Finalmente, respecto a la interacción de redes neuronales y reglas borrosas y su aplicación a la extracción de reglas dentro del marco de Computational Intelligence y Soft Computing, se puede consultar [Jang y otros, 1997; Karray y Silva, 2004; Lin y Lee, 1996; Nauck y otros, 1997].

18.7 Resumen

Los problemas a la hora de aplicar un modelo no explícito o incomprendible en muchas aplicaciones reales son numerosos. En primer lugar, es difícil contrastar el modelo con el conocimiento previo existente sobre el tema, o complementarlo con dicho conocimiento, o integrarlo en el *know-how* de la organización. En segundo lugar, no se puede explicar el porqué de cada decisión, lo que dificulta la auditoría y la transparencia de su aplicación. En tercer lugar, y como consecuencia de lo anterior, los usuarios son reacios a la aceptación del modelo. Finalmente, el procesamiento

automático y su integración en herramientas y aplicaciones informáticas de decisión es difícil. Todo esto hace que disponer de un modelo explícito y comprensible, p.ej. en forma de reglas, sea crucial en muchas aplicaciones, e imprescindible en otras, como p.ej. los entornos médicos, industriales, comerciales, etc.

En este capítulo hemos visto una serie de técnicas que, desde diferentes ámbitos, intentan convertir un modelo preexistente, pero no explícito o incomprensible, en un modelo en forma de reglas. En primer lugar hemos comentado que la representación en forma de reglas no es la única representación comprensible para los seres humanos ni asegura dicha comprensibilidad si el conjunto de reglas es muy grande, pero sí es una representación muy común y fácil de manejar. A continuación, hemos visto la aproximación más general de extracción de reglas en base a un entrenamiento de un nuevo modelo, utilizando el modelo original como oráculo para etiquetar un conjunto de datos. Esta forma de extracción de reglas se ha denominado método mimético y goza de gran flexibilidad (al poderse aplicar a cualquier tipo de modelo), facilidad de implementación y adaptación según las aplicación.

Respecto a las técnicas específicas, hemos dedicado una sección a la extracción de reglas sobre redes neuronales, técnicas que datan desde los mismos inicios del éxito de las redes neuronales, ya que los detractores de las redes neuronales siempre han utilizado el problema de la incomprensibilidad como arma arrojadiza. Del mismo modo, otras técnicas específicas de extracción de reglas se han visto a continuación.

En el último punto, se han estudiado un tipo de reglas más expresivas, basadas en los conjuntos borrosos. Los trabajos que hibridan los conjuntos de reglas borrosos, las redes neuronales y, frecuentemente, la computación evolutiva, han dado lugar a diferentes técnicas, dentro del área de “soft computing” y “computational intelligence” [Duch y otros, 2004; Mitra y otros, 2002], que permiten convertir modelos existentes en conjuntos de reglas borrosos.

En general, las diferentes técnicas introducidas en este capítulo, aunque todas ellas con una base en el aprendizaje automático, pueden ser aplicadas a diferentes ámbitos donde la IAI requiere de conocimiento que sea interpretable y procesable por los humanos y también por herramientas semiautomáticas de Ingeniería del Conocimiento. Así, las técnicas introducidas son útiles para la adquisición de conocimiento, la integración de conocimiento, la ingeniería inversa, la transformación de modelos, y muchas otras áreas donde las bases de conocimiento hayan de crearse o modificarse a partir de la experiencia (de expertos o de casos individuales).

Hemos dejado fuera otro tipo de representaciones en forma de reglas que pueden ser especialmente concisas y apropiadas en diversos ámbitos, como por ejemplo la representación relacional, utilizando lógica de primer orden. Las técnicas de extracción de reglas relacionales, originalmente bajo el paraguas denominado “Programación Lógica Inductiva” [Muggleton y Raedt, 1994], más recientemente rebautizado como “Relational Data Mining” [Dzeroski y Eds., 2001], permiten extraer modelos comprensibles a partir de datos y son, por tanto, útiles para la extracción de reglas en aplicaciones que requieren representaciones complejas: bioquímica y genética, medicina, redes, web, multimedia, lenguaje natural, etc.

18.8 Ejercicios resueltos

18.1. Sorprendentemente para los tiempos que corren, el servicio público de salud de Utópica del Norte decide incluir las operaciones de cirugía ocular para la corrección de la miopía y el astigmatismo dentro de las prestaciones básicas para la población. Se dispone de dos modelos, uno consensuado a partir de distintos expertos y otro obtenido por técnicas de aprendizaje automático.

MODELO 1:

SI Astig.=No Y $25 < \text{Edad} < 50$ Y $1.5 < \text{Miopía} < 10$ ENTONCES Operación=Sí
SI Astig.=Sí Y $25 < \text{Edad} < 50$ Y $\text{Miopía} < 6$ ENTONCES Operación=Sí
EN OTRO CASO Operación= NO

MODELO 2:

SI Edad < 30 ENTONCES Operación=NO
SI Edad > 55 ENTONCES Operación=NO
SI Miopía > 12 ENTONCES Operación=NO
SI $30 < \text{Edad} \leq 55$ y Astig. = Si y Miopía > 8 ENTONCES Operación=NO
EN OTRO CASO Operación= Sí

Según la función de complejidad de [Lu y otros, 1995] vista en la introducción de este capítulo, evalúa qué modelo basado en reglas de los anteriores es menos complejo.

Solución:

La función de [Lu y otros, 1995] mide la complejidad simplemente como $Complejidad = 2 \cdot R + C$, es decir, el número de reglas pesa el doble que el total de condiciones. El modelo 1 tiene 3 reglas y 9 condiciones (nótese que las condiciones del estilo $25 < \text{edad} < 50$ representan en realidad dos condiciones, $25 < \text{edad} \wedge \text{edad} < 50$). Así, su complejidad es $2 \cdot 3 + 9 = 15$. El modelo 2 tiene 5 reglas y 7 condiciones. Por tanto, su complejidad es $2 \cdot 5 + 7 = 17$. Por tanto, el modelo 1 es más simple, según esta función de complejidad.

■

18.2. Dado un multiclásificador aprendido sobre el conjunto de datos representados en la Figura 18.8 (40 instancias), con la correspondiente distribución de sus valores, donde podemos ver para los atributos categóricos la distribución de valores y para los numéricos el mínimo y máximo, la media, la desviación típica y el coeficiente de asimetría, ¿qué tipo de técnica de generación de datos inventados sería mejor para capturar la semántica en forma de reglas a partir del multiclásificador y pensando que se utilizará la técnica mimética con C4.5 si queremos generar 1.000 ejemplos?

Solución:

Con 1.000 ejemplos, al existir valores numéricos, es muy difícil cubrir todo el espacio de datos, pero sería recomendable respetar la distribución a priori para los atributos nominales (usando remuestreo) y utilizar una distribución normal con la media, desviación típica y asimetría estimadas a partir de los datos originales para generar los

valores numéricos. Esto si no se quiere ir a estimaciones más sofisticadas, por ejemplo, bayesianas, como el algoritmo EM.

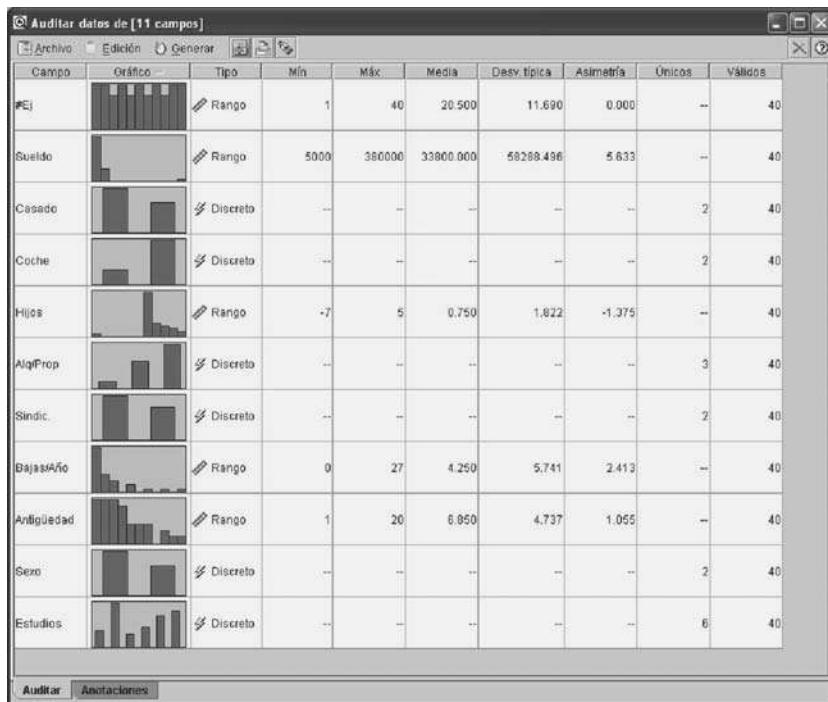


Figura 18.8: Conjunto de datos para el Ejercicio 2.

18.3. Aplicar el Algoritmo 18.3 de la sección 18.3.2 para extraer las reglas correspondientes al nodo mostrado en la Figura 18.9.

Solución:

El primer paso es construir el conjunto S_P de subconjuntos de entradas cuya suma de pesos (positivos) sea mayor que el umbral del nodo, en nuestro caso 4. Teniendo en cuenta que las entradas A , B y C son positivas, entonces $S_P = \{\{A\}, \{B, C\}, \{A, B\}, \{A, C\}, \{A, B, C\}\}$. Ahora bien, como sólo hay una entrada de peso negativo (D) el conjunto S_N sólo contiene este nodo, es decir, $S_N = \{D\}$. Así pues, para cada uno de los elementos de S_P únicamente tenemos que analizar qué pasa cuando se incluye o no la entrada D :

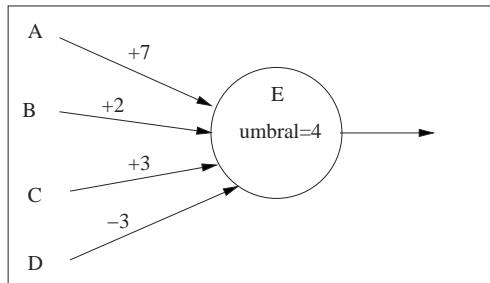


Figura 18.9: Nodo con 4 entradas y umbral de activación.

- $p = \{A\}$: Como $\text{peso}(p \cup \{D\}) = 4$ no supera el umbral mientras que $\text{peso}(p) > \text{umbral}$ entonces creamos la regla
IF (A and not D) THEN E
- $p = \{B, C\}$: Como $\text{peso}(p \cup \{D\}) = 2$ no supera el umbral mientras que $\text{peso}(p) > \text{umbral}$ entonces creamos la regla
IF (B and C and not D) THEN E
- $p = \{A, B\}$: Como $\text{peso}(p \cup \{D\}) = 6$ supera el umbral creamos la regla
IF (A and B) THEN E
- $p = \{A, C\}$: Como $\text{peso}(p \cup \{D\}) = 7$ supera el umbral creamos la regla
IF (A and C) THEN E
- $p = \{A, B, C\}$: Como $\text{peso}(p \cup \{D\}) = 9$ supera el umbral creamos la regla
IF (A and B and C) THEN E

■

18.9 Ejercicios propuestos

- 18.1.** Como ejemplo de la técnica mimética vista al principio del capítulo, descargar un conjunto de datos del repositorio UCI [Newman y otros, 1998] que tenga una salida nominal (es decir, un problema de clasificación). Separar parte del conjunto para test (30 %) y el resto para entrenamiento (70 %). Instálese una herramienta de minería de datos o de aprendizaje automático que tenga técnicas de redes neuronales y árboles de decisión, por ejemplo WEKA (en este sistema puedes usar perceptrón multicapa y J48). Entrenar una red neuronal con los datos de entrenamiento. A partir de los datos de entrenamiento generar un conjunto inventado. Utilice distintos tamaños y distintos métodos de generación de los datos inventados (uniforme, normal, remuestreo). Etiquetar, en cada caso, los datos inventados con la red neuronal. Entrenar un árbol de decisión con los datos inventados etiquetados. Comparar, en cada caso,

la precisión y el tamaño del modelo mimético (árbol de decisión) con el modelo original (red neuronal). ¿Qué técnica de generación de datos inventados y tamaño del conjunto inventados dan un mejor compromiso entre precisión y comprensibilidad?

18.2. Continuar aplicando la técnica RULENEG (Algoritmo 18.2) para extraer el conjunto de reglas correspondientes al ejemplo de la Figura 18.1 de la sección 18.3.1.

18.3. Aplicar la técnica REAL (Algoritmo 18.1) al mismo problema del ejercicio anterior, suponiendo que la función “ejemplos” del algoritmo devuelva los ejemplos pertenecientes al conjunto usado para entrenar la red. Comparar estas reglas con las obtenidas con la técnica RULENEG del ejercicio anterior.

18.4. Supóngase que el algoritmo *subset* ha generado las siguientes reglas para un nodo de una red neuronal:

```
IF a3 THEN true
IF a5 THEN true
IF (a1 and a2) THEN true
IF (a2 and a4) THEN true
IF (a1 and a4) THEN true
```

Simplificar este conjunto de reglas escribiéndolas como reglas del tipo *M-of-N*.

18.5. Comentar las ventajas que podría tener una representación basada en reglas borrosas de los modelos que aparecen en el problema resuelto 1 de la sección anterior.

Referencias

- ANDREWS, R.; DIEDERICH, J. y TICKLE, A. B.: «Survey and critique of techniques for extracting rules from trained artificial neural networks.» *Knowledge-Based Systems*, 1995, **8(6)**, pp. 373–389.
- ANDREWS, R. y GEVA, S.: «Inserting and extracting knowledge from constrained error back propagation network». En: *Proc. of the 6th Australian Conference on Neural networks*, pp. 29–32, 1995a.
- ANDREWS, R. y GEVA, S.: «Rule extraction from a constrained error back propagation MLP». En: *Proc. of the 5th Australian Conference on Neural networks*, pp. 9–12, 1995b.
- BARAKAT, N. y DIEDERICH, J.: «Learning-based rule-extraction from support vector machines». En: *The 14th International Conference on Computer Theory and applications ICCTA '2004*, , 2004.
- BARAKAT, N. y DIEDERICH, J.: «Eclectic rule-extraction from support vector machines». *International Journal of Computational Intelligence*, 2005, **2(1)**, pp. 59–62.
- BENÍTEZ, J. M.; CASTRO, J. L. y REQUENA, I.: «Are artificial neural networks black boxes». *IEEE Transactions on Neural Networks*, 1997, **8**, pp. 1156–1164.
- BLANCO-VEGA, R.: *Extracción y contextualización de reglas comprensibles a partir de modelos de caja negra*. Tesis doctoral, Departamento de Sistemas Informáticos y Computación (DSIC), Universidad Politécnica de Valencia, 2007.
- BONARINI, A.: «Evolutionary Learning of Fuzzy rules: competition and cooperation». En: W. Pedrycz (Ed.), *Fuzzy Modelling: Paradigms and Practice*, pp. 265–284. Norwell, MA: Kluwer Academic Press, 1996.
- BOZ, O.: «Extracting decision trees from trained neural networks». En: *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 456–461, 2002.
- BREIMAN, L. y SHANG, N.: «Born again trees». *Informe técnico*, Statistics Department, University of California at Berkeley, 1997.
- BRULÉ, J. y BOUNT, A.: *Knowledge Acquisition*. McGraw-Hill, 1998.
- CARSE, B.; FOGARTY, T. C. y MUNRO, A.: «Evolving fuzzy rule based controllers using genetic algorithms». *Fuzzy Sets and Systems*, 1996, **80(3)**, pp. 273–293.
- CASTRO, J. L.; MANTAS, C. J. y BENÍTEZ, J. M.: «Interpretation of artificial neural networks by means of fuzzy rules». *IEEE Transactions on Neural Networks*, 2002, **13**, pp. 101–116.

- CORDÓN, O.; DEL JESÚS, M. J. y HERRERA, F.: «Genetic learning of fuzzy rule-based classification systems co-operating with fuzzy reasoning methods». *International Journal of Intelligent Systems*, 1998, **13(10/11)**, pp. 1025–1053.
- CORDÓN, O.; GOMIDE, F. A. C.; HERRERA, F.; HOFFMANN, F. y MAGDALENA, L.: «Ten years of genetic fuzzy systems: current framework and new trends.» *Fuzzy Sets and Systems*, 2004, **141(1)**, pp. 5–31.
- CORDÓN, O. y HERRERA, F.: «A Three-Stage Evolutionary Process for Learning Descriptive and Approximative Fuzzy Logic Controller Knowledge Bases from Examples». *Journal of Approximate Reasoning*, 1997, **17(4)**, pp. 369–407.
- CORDÓN, O. y HERRERA, F.: «A two-stage evolutionary process for designing TSK fuzzy rule-based systems». *IEEE Trans. on Systems, Man, and Cybernetics*, 1999, **29(6)**, pp. 613–617.
- CORDÓN, O.; HERRERA, F.; HOFFMANN, F. y MAGDALENA, L.: *GENETIC FUZZY SYSTEMS. Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. World Scientific, 2001.
- CRAVEN, M. y SHAVLIK, J.: «Rule Extraction: Where Do We Go from Here?», 1999. Department of Computer Sciences, University of Wisconsin, Machine Learning Research Group Working Paper 99-1.
- CRAVEN, M. W. y SHAVLIK, J. W.: «Using Sampling and Queries to Extract Rules from Trained Neural Networks». En: *Proc. 11th Int'l Conf. on Machine Learning*, volumen 7, pp. 37–45. The MIT Press, 1994.
- CRAVEN, M. W. y SHAVLIK, J. W.: «Extracting Tree-Structured Representations of Trained Networks». En: *Advances in Neural Information Processing Systems*, volumen 8, pp. 24–30. The MIT Press, 1996.
- DARBARI, A.: «Rule Extraction from Trained ANN: A Survey». *Informe técnico*, Institute of Artificial Intelligence, Dept. of Computer Science, TU Dresden, Germany, 2001.
- DEBENHAM, J.: *Knowledge System Design*. Prentice Hall, 1998.
- DEMPSTER, A. P.; LAIRD, N. M. y RUBIN, D. B.: «Maximum likelihood from incomplete data via the EM algorithm». *Journal of the Royal Statistical Society, Series B*, 1977, **39(1)**, pp. 1–38.
- DOMINGOS, P.: «Knowledge acquisition from examples via multiple models». En: *Proc. 14th International Conference on Machine Learning*, pp. 98–106. Morgan Kaufmann, 1997a.
- DOMINGOS, P.: «Learning Multiple Models without Sacrificing Comprehensibility.» En: *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97*, p. 829, 1997b.

- DOMINGOS, P.: «Knowledge discovery via multiple models». *Intelligent Data Analysis*, 1998, **2**(3), pp. 187–202.
- DUCH, W.; ADAMCZAK, R. y GRABCZEWSKI, K.: «Extraction of logical rules from backpropagation networks». *Neural Processing Letters*, 1998, **7**, pp. 211–219.
- DUCH, W.; SETIONO, R. y ZURADA, J. M.: «Computational intelligence methods for rule-based data understanding». *Proceedings of the IEEE*, 2004, **92**(5), pp. 771–805.
- DZEROSKI, S. y EDs., N. LAVRAC: *Relational Data Mining*. Springer, Berlin, 2001.
- ENGELBRECHT, AP.; ROUWHORST, S. y SCHOEMAN, L.: «A Building Block Approach to Genetic Programming for Rule Discovery». En: CNewton HA Abbass, R Sarkar (Ed.), *Data Mining: A Heuristic Approach*, pp. 175–189, 2001.
- ESTRUCH, V.; FERRI, C.; HERNÁNDEZ-ORALLO, J. y RAMÍREZ-QUINTANA, M. J.: «Simple Mimetic Classifiers». En: *Proceedings of the Third International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM*, pp. 156–171, 2003.
- FEIGENBAUM, E. A.: «Some challenges and grand challenges for computational intelligence». *Journal of the ACM*, 2003, **50**, pp. 32–40.
- FU, L.: «Rule learning by searching on adapted nets». En: *Proc. of the 9th National Conference on Artificial Intelligence*, pp. 590–595, 1991.
- FUNG, G.; SANDILYA, S. y RAO, R. B.: «Rule extraction from linear support vector machines». En: *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 32–40. ACM Press, 2005.
- GALLANT, S.: *Neural network learning and expert systems*. MIT Press, 1993.
- GONZÁLEZ, A. y PÉREZ, R.: «SLAVE: A Genetic Learning System Based on an Iterative Approach». *IEEE-FS*, 1999, **7**(2), pp. 176–191.
- GREENWELL, M.: *Knowledge Engineering for Expert Systems*. Ellis Horwood Limited, 1988.
- GROVER, M. D.: «A Pragmatic Knowledge Aquisition Methodology». En: *Proc. of the 8th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 436–438. Karlsruhe, Germany, 1983.
- HERNÁNDEZ, J.; RAMÍREZ, M. J.; y FERRI, C.: *Introducción a la Minería de Datos*. Pearson Prentice Hall, 2004.
- HOFFMANN, F. y PFISTER, G.: «Evolutionary design of a fuzzy knowledge base for a mobile robot». *Int. J. of Approximate Reasoning*, 1997, **17**(4), pp. 447–469.
- ISHIBUCHI, H.; MORIOKA, K. y TÜRKSEN, B.: «Learning by fuzzified neural networks.» *Int. J. Approx. Reasoning*, 1995, **13**(4), pp. 327–358.

- ISHIBUCHI, H.; NAKASHIMA, T. y MURATA, T.: «Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems.» *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 1999, **29**(5), pp. 601–618.
- JACOBSSON, H.: «Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review». *Neural Comput.*, 2005, **17**(6), pp. 1223–1263.
- JAN, J.-S. R. y SUN, C.-T.: «Functional equivalence between radial basis function networks and fuzzy inference systems». *IEEE Trans. Neural Networks*, 1993, **4**, pp. 156–159.
- JANG, J.-S. R.; SUN, C.-T. y MIZUTANI, E.: *Neuro-Fuzzy and Soft Computing*. Prentice-Hall, 1997.
- KARRAY, F. O. y SILVA, C. W. DE: *Soft Computing and Intelligent Systems Design: Theory, Tools and Applications*. Addison Wesley, 2004.
- KOHAVI, R. y SOMMERFIELD, D.: «Targeting Business Users with Decision Table Classifiers». En: *Knowledge Discovery and Data Mining*, pp. 249–253, 1998.
- KOHAVI, R. y WOLPERT, D. H.: «Bias Plus Variance Decomposition for Zero-One Loss Functions». En: Lorenza Saitta (Ed.), *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 275–283. Morgan Kaufmann, 1996.
- KOLMAN, E. y MARGALIOT, M.: «Knowledge Extraction from Neural Networks Using the All-Permutations Fuzzy Rule Base: The LED Display Recognition Problem». En: *Proceedings of the 8th International Work-Conference on Artificial Neural Networks, IWANN*, pp. 1222–1229, 2005.
- KONONENKO, I.: «Comparison of Inductive and Naive Bayesian Learning Approaches to Automatic Knowledge Acquisition». *Current Trends in Knowledge Acquisition*, 1990, pp. 190–197.
- KOSKO, B.: *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, NJ: Prentice Hall, 1992.
- LEE, M. A. y TAKAGI, H.: «Integrating design stages of fuzzy systems using genetic algorithms». En: *Proceedings of the Second IEEE International Conference on Fuzzy Systems, FUZZ-IEEE'93*, pp. 613–617, 1993.
- LI, M. y VITÁNYI, P.: *An Introduction to Kolmogorov Complexity and its Applications*. 2nd Ed. Springer-Verlag, 1997.
- LIN, C.-T. y LEE, C.S.G.: *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Upper Saddle River, NJ: Prentice Hall, 1996.
- LU, H.; SETIONO, R. y LIU, H.: «Towards Effective Classification Rule Extraction (Abstract).» En: *Proceedings of the DOOD'95 Post-Conference Workshops on Integration of Knowledge Discovery in Databases with Deductive and Object-Oriented Databases (KDOOD) and Temporal Reasoning in Deductive and Object-Oriented Databases (TDOOD)*, pp. 47–49, 1995.

- MAGDALENA, L. y MONASTERIO-HUELIN, F.: «A Fuzzy Logic Controller with Learning through the Evolution of its Knowledge Base». *International Journal of Approximate Reasoning*, 1997, **16(3/4)**, pp. 335–358.
- MARTÍNEZ, R. GARCÍA: *Adquisición de Conocimiento*. En Abecasis, S. y Heras, C. *Metodología de la Investigación*. Editorial Nueva Librería., 1994.
- McMILLAN, C.; MOZER, M. C. y SMOLENKY, P.: «Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network». En: *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, , 1991.
- MEYER, M. y BOOKER, J.: *Eliciting and analyzing expert judgment: a practical guide*. Society for Industrial and Applied Mathematics, 2001.
- MIKA, S.; RATSCH, G.; WESTON, J.; SCHOLKOPF, B. y MULLER, K.: En: *Proceedings of IEEE Neural Networks for Signal Processing Workshop*, 8 pages, , 1999.
- MITRA, S.; PAL, S. K. y MITRA, P.: «Data mining in soft computing framework: a survey». *IEEE Trans. Neural Networks*, 2002, **13**, pp. 3–14.
- MUGGLETON, S. y RAEDT, L. DE: «Inductive Logic Programming: Theory and Methods». *Journal of Logic Programming*, 1994, **19/20**, pp. 629–679.
- NAUCK, D.; KLAWOON, F. y KRUSE, R.: *Foundations of Neuro-Fuzzy Systems*. John Wiley & Sons, 1997.
- NEWMAN, D. J.; HETTICH, S.; BLAKE, C. L. y MERZ, C. J.: «UCI Repository of machine learning databases», 1998. [Http://www.ics.uci.edu/~mlearn/MLRepository.html](http://www.ics.uci.edu/~mlearn/MLRepository.html)
- NÚÑEZ, H.; ANGULO, C. y CATALÀ, A.: «Rule extraction from support vector machines.» En: *Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN*, pp. 107–112, 2002.
- PARK, D.; KANDEL, A. y LANGHOLZ, G.: «Genetic-based new fuzzy reasoning models with application to fuzzy control». *IEEE Transactions on Systems, Man, and Cybernetics*, 1994, **24(1)**, pp. 39–47.
- PARODI, A. y BONELLI, P.: «A New Approach to Fuzzy Classifier Systems». En: *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 223–230. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
- PAZZANI, M. J.: «Influence of Prior Knowledge on Concept Acquisition: Experimental and Computational Results». *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 1991, **17(3)**, pp. 416–432.
- PHAM, D. T. y KARABOGA, D.: «Optimum design of fuzzy logic controllers using genetic algorithms». *Journal of Systems Engineering*, 1991, **1**, pp. 114–118.

- POP, E.; HAYWARD, R. y DIEDERICH, J.: «RULENEG: Extracting rules from a trained ANN by stepwise negation». *Informe técnico*, Neurocomputing Research Centre Tech. Rep., Queensland University of Technology, 1994.
- RISSANEN, J.: «Modelling by shortest data description». *Automatica*, 1978, **14**, pp. 465–471.
- SAITO, K. y NAKANO, R.: «Medical diagnostic expert system based on PDP model». En: *Proceedings of IEEE International Conference on Neural Networks*, volumen 1, pp. 255 – 262, 1988.
- SESTITO, S. y DILLON, T. S.: *Automated knowledge acquisition*. Prentice-Hall, Inc., 1994.
- SHAVLIK, J. y TOWELL, G.: «Knowledge-Based Artificial Neural Networks». *Artificial Intelligence*, 1994, **70(1,2)**, pp. 119–165.
- TER BOR, R.: «Extracting fuzzy rules using genetic programming author: Rutger ter Borg». *Informe técnico*, Master of Science thesis Delft University of Technology, 2001.
- THRIFT, P. R.: «Fuzzy Logic Synthesis with Genetic Algorithms.» En: *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 509–513, 1991.
- THRUN, S.: «Extracting Rules from Artificial Neural Networks with Distributed Representations». En: *Advances in Neural Information Processing Systems*, volumen 7, pp. 505–512. The MIT Press, 1995.
- TICKLE, A. B.; ANDREWS, R.; GOLEA, M. y DIEDERICH, J.: «The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks». *IEEE Transactions on Neural Networks*, 1998, **9(6)**, pp. 1057–1068.
- TICKLE, A. B.; ORLOWSKI, M. y DIEDERICH, J.: «DEDEC: a methodology for extracting rule from trained artificial neural networks». En: *Proceedings of the AISB'96 Workshop on Rule Extraction from Trained Neural Networks*, pp. 90–102, 1996.
- TOWELL, G. y SHAVLIK, J.: «The Extraction of Refined Rules from Knowledge-Based Neural Networks». *Machine Learning*, 1993, **13(1)**, pp. 71–101.
- TUNSTEL, E. y JAMSHIDI, M.: «On Genetic Programming of Fuzzy Rule-Based Systems for Intelligent Control». *International Journal of Intelligent Automation and Soft Computing*, 1996, **2(3)**, pp. 273–284.
- VALENZUELA-RENDÓN, M.: «The Fuzzy Classifier System: Motivations and first Results». En: *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pp. 338–342. Springer-Verlag, 1991.

- VELASCO, J. R.: «Genetic-based on-line learning for fuzzy process control». *International Journal of Intelligent Systems*, 1998, **13(0-11)**, pp. 891–903.
- WALLACE, C. S. y BOULTON, D. M.: «An Information Measure for Classification». *Computer Journal*, 1968, **11(2)**, pp. 185–194.
- WALLACE, C. S. y DOWE, D. L.: «Minimum Message Length and Kolmogorov Complexity». *The Computer Journal*, 1999, **42(4)**, pp. 270–283.
- ZHOU, Z.-H. y JIANG, Y.: «Medical Diagnosis with C4.5 Rule Preceded by Artificial Neural Network Ensemble», 2003.
- ZHOU, Z.-H. y JIANG, Y.: «NeC4.5: Neural Ensemble Based C4.5», 2004.
- ZHOU, Z.-H.; JIANG, Y. y CHEN, S.-F.: «Extracting Symbolic Rules from Trained Neural Network Ensembles», 2003.

Parte VI

ASPECTOS
METODOLÓGICOS Y
APLICACIONES

Capítulo 19

Ingeniería del Conocimiento

Adolfo Lozano Tello
Universidad Extremadura

19.1 Introducción a la Ingeniería del Conocimiento

La Ingeniería del Conocimiento (IC) es la disciplina de la IA que proporciona los métodos y técnicas para construir sistemas computacionales denominados Sistemas Basados en Conocimiento (SBCs). Estos sistemas se diferencian de otros sistemas inteligentes porque manejan una gran cantidad de conocimiento de un dominio para resolver alguna tarea.

El término “conocimiento” no es fácil de definir, ni tampoco de distinguir de otros como “dato” o “información” [Davenport y Prusak, 1999], aunque en la vida diaria se usen indiscriminadamente. Se puede decir que un dato es una representación de un valor, un concepto o un hecho preparado para su uso por un agente humano o artificial. Así, el valor de un sensor de temperatura puede considerarse como dato si está almacenado en algún soporte físico o puede ser percibido por algún agente. La información consiste en asignar un significado a un dato para identificarlo y describirlo con el propósito de ser utilizado por un agente. Esta definición, aunque sutil, permite diferenciar el dato de la información por la forma en la que los humanos dotamos de semántica a ciertos elementos preparándolos para realizar alguna actividad. Por ejemplo, constituye información la afirmación: “Si el valor del sensor es mayor que 80° , existe peligro”. En este caso, la información asociada a un sensor y relacionada con un dato (el valor del sensor) está disponible y preparada para ser explotada. El conocimiento se basa en estos dos términos y se puede definir como la capacidad de utilizar los datos e información con el objetivo de resolver alguna tarea. Si un sistema posee conocimiento, sabrá cómo seleccionar y manejar la información y los datos que se consideren relevantes en ese momento para resolver el problema para el que fue diseñado. Por ejemplo, un sistema con conocimiento sabrá qué acciones realizar para evitar una situación de peligro cuando el valor del sensor supere los 80° . Aun teniendo en cuenta estas definiciones, poder diferenciar un elemento como dato, información o conocimiento no es trivial, ya que la frontera entre ellos es muy débil.

Un agente que disponga de conocimiento, para obtener una salida o tomar una decisión, debe saber escoger qué método aplicar para resolver un problema determinado, qué información y datos son relevantes y en qué grado y, además, producir nuevos datos e información porque sean necesarios en el proceso de resolución. Así, para el caso concreto de los sistemas software que usan conocimiento, su diseño puede tener en cuenta aspectos probabilísticos, la precisión de la salida frente a tiempos de respuesta, los riesgos asumidos en la toma de decisiones y otros elementos configurables o adaptables según el caso del problema. Además, estos sistemas pueden seleccionar los datos o hechos importantes y decidir qué información es útil en cada caso para alcanzar su objetivo. Es decir, de entre todos los datos e información disponibles, pueden seleccionar la información importante en ese momento, ponderarla y dar prioridad a unos elementos frente a otros. Por otro lado, otra cualidad significativa de los sistemas con conocimiento es su capacidad para inferir nueva información a partir de la que almacenan explícitamente. Disponen de módulos de razonamiento para poder deducir hechos a partir de otros hechos y reglas existentes en el sistema y, así, llegar a resultados a partir de procesos de inferencia. También, algunos pueden justificar la solución alcanzada mostrando, junto a las salidas, los pasos seguidos en el proceso de resolución.

Cuando se empezaron a diseñar los primeros SBCs que poseían las anteriores cualidades, la manera de desarrollarlos era sobre todo intuitiva y, a menudo, se basaba en métodos de prueba y error. No existían directrices ni técnicas para su construcción y, en el mejor de los casos, se usaban procesos de Ingeniería del Software (IS) diseñados para sistemas de información. La IC surge para paliar estas carencias y, aunque basada en la IS, adecúa sus procesos de construcción a las cualidades propias de los SBCs. Se puede decir que la IC establece el conjunto de guías, métodos y técnicas para desarrollar de manera formal sistemas capaces de resolver problemas, en cierta forma complejos, que no aborda la IS tradicional y cuyo principal recurso es el tratamiento del conocimiento. Se debe hacer notar que, aunque la IC ha tomado muchas de las técnicas existentes en IS, los nuevos métodos de IS también incorporan algunas de las diseñadas específicamente para el uso del conocimiento. A grandes rasgos, se puede decir que la IC trata de transformar el conocimiento humano en conocimiento utilizable por sistemas artificiales para resolver un problema. Para realizar esta transformación, en la mayoría de las metodologías de IC se identifican las siguientes fases:

- **Identificación del problema.** Consiste en estudiar la tarea que debe resolver el SBC, analizando si es viable técnicamente y si es oportuno según las condiciones actuales del entorno. En esta fase se debe decidir si el problema es adecuado para ser tratado por un SBC, ya que puede que el problema sea demasiado simple, demasiado complejo, demasiado caro o no se den las condiciones adecuadas para ser abordado por la IC.
- **Adquisición de conocimientos.** Consiste en recopilar conocimientos desde expertos u otras fuentes de conocimientos. Este es uno de los procesos que más tiempo conlleva en la construcción del sistema, y debe realizarse en todo el ciclo de vida del SBC.

- **Conceptualización:** consiste en estructurar un modelo conceptual que va a componer el sistema de forma descriptiva en algún sistema de representación. En esta fase se organiza y describe el conocimiento adquirido de forma que pueda ser revisado y evaluado por las personas involucradas en el desarrollo del SBC.
- **Formalización:** es una representación semicomputable del modelo conceptual de forma que permita realizar las deducciones necesarias para poder interpretar los conocimientos almacenados. Como se vio en la parte II, el Ingeniero del Conocimiento (InCo) deberá decidir qué formalismos de representación e inferencia deben usarse en el SBC.
- **Implementación del sistema:** consiste en codificar los modelos diseñados en un modelo computable y preparar el sistema para que puedan incorporarse modificaciones, prestando especial atención a la actualización del conocimiento.
- **Evaluación:** es el conjunto de actividades que se llevan a cabo durante toda la fase de desarrollo del SBC para asegurar la fiabilidad y calidad del sistema.

A continuación, en la sección 19.2, se presentan algunas técnicas usadas en la fase de adquisición del conocimiento para la construcción de SBCs. En la sección 19.3 se definen los SBCs, y se describen su estructura y características, comparándolos con los sistemas de información tradicionales. Seguidamente, en la sección 19.4, se realiza un breve repaso histórico de los métodos de desarrollo de SBCs. El resto de este capítulo, que comprende la sección 19.5, estudia con más detalle la metodología CommonKADS, utilizando en cada uno de los apartados un ejemplo de construcción de un SBC.

19.2 Adquisición del conocimiento

Uno de los procesos más costosos que debe realizarse en el desarrollo de SBCs es la adquisición del conocimiento. Constituye un auténtico cuello de botella que conlleva un enorme esfuerzo de tiempo y recursos, pero la rigurosidad con la que se aborde esta fase va a repercutir directamente en la calidad del SBC. Consiste en ir recopilando el conocimiento que va a necesitar el sistema antes de la conceptualización. Simplificando mucho, se puede decir que la adquisición del conocimiento es el “trabajo de campo” donde se recoge el vocabulario del dominio, los casos de uso, la forma de proceder, la forma de razonar, los casos excepcionales, la prioridad de los factores de decisión y demás elementos que van a servir al sistema para dar una salida. De este modo, la conceptualización puede verse como el “trabajo de oficina” donde se analiza, organiza, describe y estructura todo este contenido de una manera más formal, de forma que pueda supervisarse y evaluarse por las personas involucradas en el proyecto. La conceptualización no tiene por qué esperar a que termine la fase de adquisición del conocimiento, sino que lo habitual es que se vayan completando ambas de forma incremental. Es decir, a medida que se va obteniendo el conocimiento de un dominio, se puede ir modelando hasta llegar a un grado adecuado para representarlo.

En muchos dominios, el proceso de adquisición y conceptualización debe aplicarse en todo la vida del sistema, aunque el trabajo mayor se necesita hacer en las fases iniciales de desarrollo del sistema.

Aunque las fuentes de conocimiento que contienen el conocimiento humano son muy variadas: libros, manuales, publicaciones científicas, presentaciones, informes, partes de trabajo, bases de datos, *blogs* de Internet, y muchos otros, la mayoría de las veces es necesario educir el conocimiento a partir de expertos en la materia que trata el dominio del SBC. Los expertos mantienen un conocimiento muy extenso y bien relacionado sobre su dominio, conocen las reglas que permiten descubrir nuevas hipótesis, son capaces de valorar los riesgos y probabilidades de las decisiones y son capaces de dar solución a un gran número de problemas basándose, sobre todo, en experiencias previas obtenidas en casos pasados. En la adquisición, el experto debe transmitir al Ingeniero de Conocimiento (InCo) el vocabulario con el que trabaja y la forma de resolver los problemas habituales. Esta tarea conlleva importantes problemas debido a que el experto basa su experiencia a situaciones resueltas anteriormente pero que, a menudo, no puede explicar directamente cómo llegó a esos resultados. Es más, cuanto mayor es la experiencia del experto, más difícil le es razonar cómo dio una respuesta a un caso determinado. El objetivo del InCo es buscar los factores y procedimientos que han hecho razonar al experto hasta llegar a esa solución.

Se debe hacer notar que el proceso de educación del conocimiento desde el experto puede encontrarse con muchos problemas, no sólo por la dificultad intrínseca de la transmisión de resolución de casos complejos, sino porque el experto puede ser reacio a ceder su sabiduría a otros. Es lógico que llegue a pensar que puede perder su estatus de importancia en su dominio o incluso que puede ser prescindible en su puesto de trabajo. El modo de colaboración del experto debería estar negociado a priori antes de comenzar el proceso de adquisición.

Existen varias técnicas para adquirir el conocimiento de un dominio. Siguiendo el esquema que aparece en [Alonso y otros, 2004], a continuación se comentan muy brevemente estas técnicas.

19.2.1 Técnicas manuales de adquisición del conocimiento

19.2.1.1 Entrevistas

Las entrevistas del InCo con el experto es el método más común de educir conocimiento. En las entrevistas, el experto puede contar cómo están organizados los elementos relevantes en los problemas y el proceso seguido hasta su resolución. Debe tenerse en cuenta que en las entrevistas no tiene por qué realizarse únicamente entre un InCo y un experto, sino que varios InCos y varios expertos pueden intervenir en el proceso. Los tipos de entrevistas son los siguientes:

- **Entrevista inicial:** en el primer encuentro, se debe intentar crear un buen clima entre InCo y experto para facilitar los posteriores encuentros. El contenido de la reunión puede ser: presentaciones, explicación sobre los objetivos del SBC, discusión sobre el papel del experto en el proyecto, recomendaciones de otras fuentes de conocimiento para el InCo, y fechas de las próximas reuniones.

- **Sesiones de adquisición de conocimiento general o entrevistas no estructuradas:** los objetivos de estas entrevistas son conocer los principios generales del dominio, adquirir la terminología básica y el alcance de la dificultad de resolución. En estas entrevistas se realizan preguntas del tipo: ¿Cómo resuelves este caso? ¿Qué elementos necesitas saber para tomar la decisión? ¿Cómo puedes justificar la solución que has dado? El experto podrá responder con amplitud a cada una de las cuestiones. En las entrevistas abiertas, aunque el InCo tiene el objetivo de obtener estas ideas generales, puede interactuar intercalando preguntas que clarifiquen el vocabulario o métodos de resolución.
- **Sesiones de adquisición de conocimiento específico o entrevistas estructuradas:** en este tipo de entrevistas, la agenda de los temas a tratar está más detallada por parte del InCo y las sesiones tienen unos objetivos de adquisición muy definidos. Se centran en términos o casos concretos recopilando toda la información necesaria antes de pasar al siguiente. Se realizan preguntas como: ¿Qué valores puede tomar ese concepto? ¿De qué se diferencia de otros? En este caso concreto ¿qué métodos de resolución emplearías? Las respuestas del experto serán en estas sesiones mucho más concretas que en las de adquisición de conocimiento general.

19.2.1.2 Cuestionarios

Es una variante de las entrevistas estructuradas en la que se ofrece al experto un cuestionario de preguntas concretas o fichas en las que debe completar la información. Son adecuados para completar atributos de conceptos o para pedir estimaciones de certeza o verosimilitud en una escala lingüística de valores. El problema es que requiere de conocimiento previo del dominio y un considerable esfuerzo de preparación por parte del InCo pero, en cambio, permite que el experto reflexione sobre sus respuestas, quizás modificando opiniones dadas en preguntas previas.

19.2.1.3 Extracción mediante curvas cerradas

Es utilizada cuando el dominio contiene elementos visuales que pueden ser catalogados por el experto. Consiste en pedir al experto que encierre en un círculo los componentes que intervienen en el caso para dar una determinada salida o en tomar una decisión. Por ejemplo, en un dominio de extinción forestal de incendios, usando fotos aéreas se puede pedir al experto que encierre en una curva las zonas que tienen peligro de propagación del incendio. Es una técnica muy intuitiva para el experto.

19.2.1.4 Análisis de protocolos

En el análisis de protocolos, se graban los pensamientos que va comentando en voz alta el experto conforme resuelve un problema típico de su dominio. Después, esta grabación se analiza intentando averiguar los pasos de resolución que va tomando el experto. Es un método muy útil para adquirir conocimiento sobre los métodos de razonamiento aplicado pero es costoso por el tiempo que conlleva y, para obtener

las reglas que utiliza el experto de forma general, es necesario aplicarlo varias veces. El análisis de protocolos tiene cuatro fases:

1. *Obtención del protocolo*: tras elegir un caso y dar al experto las instrucciones de cómo debe verbalizar sus pensamientos conforme resuelve el problema, se realiza la grabación.
2. *Transcripción y segmentación del protocolo*: escuchando la grabación se interpreta el monólogo del experto con frases completas con sentido. Se añaden las observaciones sobre el significado intrínseco de las frases y del comportamiento del experto en el proceso de resolución.
3. *Codificación del protocolo*: desde la transcripción, se identifican los conceptos que intervienen, sus atributos, sus relaciones y las reglas de razonamiento que ha ido indicando el experto en la secuencia de resolución.
4. *Interpretación*: se intenta recoger la estrategia de resolución del experto, las reglas de inferencia que elige en cada caso, por qué da prioridad a algunas, por qué descarta otras; en definitiva, para adquirir el método de resolución completo generalizándolo a partir de este ejemplo.

19.2.1.5 Observación del experto

Consiste en que el InCo observa al experto en condiciones reales, intentando siempre no perturbar su trabajo. El InCo tomará notas de las operaciones que realiza el experto y, posteriormente, puede preguntar al experto por qué o cómo ha realizado cada una de las acciones. Requiere tener un conocimiento previo de los términos del dominio ya que sólo puede emplearse para adquirir conocimiento del tipo procedimental.

19.2.2 Técnicas semiautomáticas de adquisición del conocimiento

19.2.2.1 Escalado psicológico

Estas técnicas sirven para producir representaciones estructuradas del conocimiento a partir de juicios realizados por expertos. Como aparece en la Tabla 19.1, en una matriz simétrica se representan en las filas y columnas todos los elementos que van a ser analizados (supongamos por ejemplo que estos elementos son tipos de minerales y se busca una taxonomía geológica). Las celdas de la matriz contendrán la distancia d_{ij} que el experto percibe que existe entre dos elementos, siguiendo alguna escala fijada previamente. Al ser una matriz simétrica, d_{ji} debe ser igual a d_{ij} , por lo que sólo es necesario completar una de las dos triangulares. En cambio, se le puede pedir al experto que complete ambas para ratificar que su juicio es acertado.

La distancia d_{ji} entre dos elementos se interpretará como la proximidad en la representación del conocimiento. Con estas apreciaciones realizadas por el experto,

se aplica algún algoritmo implementado en alguna herramienta software para obtener una representación de los elementos analizados. Dos de estos algoritmos son los siguientes:

- **Escalado multidimensional:** con las distancias de la matriz, se realiza el cálculo EMD [Barbero, 1993] en el que, para varias dimensiones, se obtiene la proyección de cada elemento en cada dimensión. Con los valores de estas proyecciones se pueden realizar agrupamientos de similitud de elementos conforme a la dimensiones obtenidas (para una visión más detallada de este método véase el capítulo 23).
- **Análisis de agrupamientos:** consiste en coger de forma consecutiva parejas de elementos (filas y columnas) con la menor distancia. Las dos filas (y columnas) de los elementos agrupados se transforman en una que contendrá la distancia mínima (u otro criterio) de los dos elementos agrupados con los demás elementos. El resultado será una jerarquía de conceptos que vendrá supeditado al orden de agrupamiento hecho en el algoritmo.

	E_1	E_2		E_n
E_1	0	d_{12}		d_{1n}
E_2	d_{21}	0		d_{2n}
.			0	
E_n	d_{n1}	d_{n2}		0

Tabla 19.1: Estructura de la matriz usada en escalado psicológico.

19.2.2.2 Emparrillado

En el emparrillado, el experto debe dar valores a características de los elementos del dominio que quieren ser evaluados. El rango de valores debe fijarse entre 1 y n , de forma que 1 sea el valor de un polo y n el opuesto. Por ejemplo, como vemos en la Tabla 19.2, tenemos ocho características bipolares (descuidado/cuidado, desconocido/famoso...) para valorar la categoría de siete campos de golf (G_i).

Estas valoraciones permiten que se puedan extraer del experto las relaciones entre elementos y las prioridades de las características, usando alguna aplicación software que realice los cálculos. A partir de esta matriz, se pueden obtener dos identificaciones: de elementos y de características.

1. **Identificación de elementos:** Para extraer los elementos desde la parrilla, se construye una nueva matriz con las distancias entre los elementos (el sumatorio de las distancias de las características) y, desde esta, se van haciendo agrupaciones jerárquicas hasta obtener un árbol ordenado de los elementos de la parrilla. Si dos elementos aparecen muy cercanos y el experto no está de acuerdo, se pide al experto que indique qué característica los hace diferentes del resto, y esta

característica se añade a la parrilla. Si dos elementos aparecen muy alejados y el experto no está de acuerdo, el experto debe decirnos qué característica los acerca y se añade a la parrilla.

2. **Identificación de características:** En este caso, se crea una nueva matriz formada por las distancias entre las características y, de igual forma que en el caso anterior, se realizan agrupaciones jerárquicas entre características. Se debe tener en cuenta que las características son bipolares y puede que el experto no haya tenido en cuenta el valor positivo del término, por lo que habría que normalizar. El árbol jerárquico de características será analizado por el experto e indicará si está de acuerdo con las cercanías de las características. Si no es así, debería buscarse algún elemento que contradiga esa cercanía.

(rango 1-5)	G1	G2	G3	G4	G5	G6	G7
descuidado/cuidado	3	2	4	5	1	1	3
desconocido/famoso	4	1	5	5	2	3	2
ruidoso/silencioso	1	2	1	4	2	1	2
caro/barato (cuota)	3	4	3	1	3	4	2
clima nefasto/clima excelente	4	4	4	2	1	2	3
mal servicio/buen servicio (empleados)	5	5	3	1	4	2	2
desequilibrado/equilibrado (hoyos)	3	3	5	1	3	2	4
saturado/deserto (jugadores)	2	1	5	3	3	2	3

Tabla 19.2: Ejemplo de valoración usada en la técnica de emparrillado.

Con la técnica del emparrillado, se pueden obtener las correlaciones entre características, analizar las que son importantes o, por otro lado, realizar una selección de los elementos más idóneos dando pesos de importancia a las características. El inconveniente mayor es la subjetividad del juicio de asignación de valores a los elementos y la dificultad de identificar las características que deben intervenir en el proceso.

19.2.3 Técnicas automáticas de adquisición del conocimiento

Existen herramientas software que ayudan a realizar clasificaciones de elementos a partir de un conjunto de ejemplos, formado por ejemplos positivos de pertenencia a una clase o que hacen verdadero un hecho, y por ejemplos negativos que serían los ejemplos que no pertenecen a la clase por no coincidir alguna característica. Mediante el conjunto de ejemplos, se analizan los atributos y valores importantes para caracterizar mediante la técnica de la inducción al objeto general o a una hipótesis. La selección del conjunto de ejemplos, o ejemplo de entrenamiento, es muy importante ya que va a determinar de forma categórica la descripción general del concepto o de la hipótesis encontrada.

Entre los más conocidos se encuentra el algoritmo W de Winston [Winston, 1970] que intenta describir las características de un concepto por inducción. La idea consiste en seleccionar ejemplos positivos, o de pertenencia completa a una clase, y ejemplos

negativos o “casiejemplos”, que son aquellos que, aunque su descripción está cerca de la clase, difieren en alguna propiedad. Con estos ejemplos, el algoritmo va formando un modelo en evolución con las características que abarcan a los ejemplos positivos y no a los negativos. De manera similar, el espacio de versiones de Mitchell [Mitchell, 1982] utiliza también ejemplos positivos y negativos sobre un hecho y va formando hipótesis genéricas y específicas recogiendo los hechos o propiedades que aparecen en los ejemplos positivos y que no se cumplen en los negativos. Otra técnica consiste en construir árboles de decisión [Quinlan, 1986] que toman como entradas objetos o situaciones positivos y negativos caracterizados mediante un conjunto de propiedades (véase el capítulo 17). El algoritmo evalúa las características y selecciona de forma progresiva aquella que mejor categoriza a los ejemplos; el resultado es un árbol que, al recibir un nuevo ejemplo, es capaz de catalogarlo como positivo o negativo. Junto a estos métodos se deben mencionar las técnicas de aprendizaje de redes neuronales vistas en el módulo anterior del libro. Estas permiten mejorar sus salidas en la clasificación de elementos tras periodos de entrenamiento. El problema para la IC es que no permite fácilmente justificar porqué se ha modificado la ponderación de las propiedades para incluir este conocimiento en el sistema.

19.2.4 Adquisición del conocimiento a partir de un grupo de expertos

A pesar de que al aumentar el número de expertos que opinan sobre su dominio se complica el proceso de adquisición, se enriquece mucho la identificación de elementos, el grado de detalle de los atributos y los puntos de vistas de resolución. Las técnicas descritas anteriormente pueden usarse con varios expertos a la vez, quizás haciendo que los expertos acuerden los valores para sus opiniones o quizás aplicando fórmulas estadísticas a todas las valoraciones de los expertos. En cambio, existen algunas técnicas especialmente pensadas para ser utilizadas en un grupo de expertos, que se describen muy brevemente a continuación:

19.2.4.1 Tormenta de ideas

La tormenta de ideas o brainstorming [Osborn, 1953], consiste en que el equipo de expertos debe exponer en voz alta los puntos de vista que tienen sobre el problema, aunque las afirmaciones no estén muy meditadas o la forma de resolución no sea la más acertada. En las primeras fases es más importante la cantidad que la calidad de las apreciaciones, por lo que se debe animar al grupo a que proponga nuevas ideas, las modifique o las fusione sin ninguna censura. El objetivo es que las ideas de uno inspiren a los demás y a sí mismo y que, por discusión, se vayan recogiendo el mayor número de puntos de vista. Un moderador debe ir recopilando todas las afirmaciones y, posteriormente, se analizan y se detalla lo que se haya acordado.

19.2.4.2 Técnica nominal de grupo

Aunque el objetivo es similar a la tormenta de ideas, con la técnica nominal de grupo [Gómez y otros, 1997] se ordena el debate y se establecen los criterios para seleccionar una u otra solución. En primer lugar, se generan por escrito las ideas, se ponen en común, se defienden y discuten y, por último, se procede a una votación de las soluciones.

19.2.4.3 Método Delphi

El método ideado por Linstone [Linstone y Turoff, 1987] propone que los expertos completen de forma anónima y por fases cuestionarios que se irán retroalimentando con las opiniones de los demás. Las fases son las siguientes: 1) Se envía el cuestionario a los expertos que deben completar en un tiempo determinado; 2) Se vuelven a enviar los cuestionarios donde se recogen las valoraciones de los demás, junto a la situación de cada valoración en cuartiles. Se sugiere a los que estén fuera de los cuartiles que modifiquen su opinión; 3) Si los que están fuera de los cuartiles mantienen la opinión deben justificar su valoración; 4) Se solicita que de nuevo hagan el cuestionario, teniendo en cuenta todas las opiniones anteriores y discutiendo sobre las discrepancias; y 5) Se obtienen los resultados finales del estudio, recogiendo las opiniones en las valoraciones discordantes. Al existir anonimato en las valoraciones, los expertos defienden con tranquilidad sus posturas y favorece los cambios de opinión y la convergencia de valoraciones.

En la fase de adquisición del conocimiento, lo ideal sería utilizar el mayor número de las anteriores técnicas para poder completar y contrastar el conocimiento. Como se puede intuir, las técnicas que se emplearán van a depender de la disponibilidad de las fuentes de conocimiento y colaboración de los expertos. Además, cuando se posee un tiempo limitado para realizar esta fase, no sólo es complicado seleccionar qué técnicas emplear, sino cómo diseñarlas y ejecutarlas. A menudo, la experiencia y pericia del InCo va a determinar la calidad en la consecución de esta tarea.

Debe indicarse que, al igual que ocurre con los proyectos de desarrollo colaborativo sobre software libre, actualmente existe gran cantidad de conocimiento representado en repositorios de ontologías (véase el capítulo 5) que está preparado para poder reutilizarse, en muchos casos de forma gratuita, en nuevos SBCs. De esta forma, un InCo que vaya a abordar el desarrollo de un sistema, deberá analizar estos repositorios y, si tiene la suerte de encontrar ontologías del dominio de su sistema, quizás únicamente tenga que especializar este conocimiento o integrar el representado en varias ontologías.

19.3 Sistemas basados en conocimiento

Las aplicaciones software que resuelven problemas complejos que no son abordables por la IS tradicional se denominan agentes o programas inteligentes. Los SBCs son agentes inteligentes que se encargan de resolver alguna tarea usando, como principal recurso, el conocimiento. Estos suelen usar procedimientos para inferir nuevo

conocimiento, incluso pueden incorporar métodos heurísticos más o menos complejos para resolver un problema, pero la diferencia con otros agentes inteligentes es que utilizan gran cantidad de conocimiento representado explícitamente. Es decir, un agente inteligente que no podría etiquetarse como SBC puede tener implementado un proceso de búsqueda de solución muy eficaz y usar poca cantidad de conocimiento, como por ejemplo un programa de ajedrez que tuviera implementado el algoritmo alfa-beta (véase el capítulo 9) que usara únicamente el conocimiento de las características de las piezas y una función simple de valoración del tablero.

A menudo se emplea también el término “sistemas expertos” para referir a los SBCs que emulan el comportamiento de expertos humanos en algún dominio concreto. En realidad, como la mayoría de las fuentes de conocimiento en la adquisición proviene de expertos humanos más que extraerlos de libros, manuales u otras fuentes, se usa indistintamente el término SBC o sistema experto, aunque hay que decir que este último está en desuso y se emplea para los sistemas que tratan de imitar las soluciones dadas por expertos a un determinado problema.

A pesar de las anteriores definiciones, es difícil poner límite en lo que se considera un sistema software tradicional, un agente inteligente, un SBC o un sistema experto. Las metodologías de IS actuales incorporan descripciones y jerarquías de clases, cercanas a las usadas en la IC y, a menudo, integran reglas heurísticas y métodos básicos de deducción para resolver algún cálculo. Por otro lado, es complicado no catalogar a un agente inteligente como SBC, debido a que no se puede definir bien qué es una gran cantidad de conocimiento. Y con respecto al término “sistema experto”, es difícil buscar un ejemplo de un SBC que no adquiera sus conocimientos principalmente de los expertos humanos porque hay que tener en cuenta que otras fuentes de conocimiento son habitualmente generadas también por personas.

19.3.1 Estructura de los SBCs

Los sistemas de información tradicionales, usando algoritmos codificados en un programa, proporcionan una información de salida a partir de los datos almacenados en el sistema. El código del programa se implementa de forma que se referencia directamente a la estructura y el tipo de datos que se establece para el sistema. En cambio, en los SBCs, existe una separación entre la información necesaria para resolver un problema y el método para resolverlo. Estos dos componentes son los elementos principales que forman el SBC, denominados base de conocimiento y motor de inferencia. Las bases de conocimiento contendrán, usando algunos de los formalismos de representación vistos en la parte II del libro, los conocimientos sobre los dominios que va a necesitar el SBC. Estos conocimientos pueden estar formados por elementos declarativos como datos o hechos, y elementos heurísticos como reglas, métodos procedimentales o de decisión. El otro componente, el motor de inferencia, explotará estas bases de conocimiento para obtener una salida. En el motor de inferencia está implementado el método de resolución que, usando los hechos y reglas de las bases de conocimientos, tendrá en cuenta estrategias, utilidades, prioridades, probabilidades y riesgos para obtener una salida óptima. Con estos elementos irá decidiendo en cada paso de resolución qué reglas y hechos deben usarse para proporcionar una solución o tomar una determinada decisión.

Además de estos dos elementos, los SBC suelen poseer interfaces de entrada y salida especiales para comunicarse con los usuarios, con dispositivos físicos o con otros SBCs. Estos módulos pueden tener implementados elementos que expliquen la línea de razonamiento que ha seguido el motor de inferencia, las probabilidades manejadas en cada deducción, o justificar por qué se ha tomado una u otra decisión en cada paso de la resolución. Además, algunos SBCs incorporan interfaces de aprendizaje que permiten incorporar conocimiento a partir de las indicaciones de los usuarios o directamente retroalimentándose de sus salidas. A nivel físico, algunos SBCs específicos pueden utilizar módulos de tratamiento de imágenes, reconocedores y sintetizadores de voz, o sensores para comunicación de entrada y salida con el medio.

Debe indicarse que, en la última década, la tendencia [Neches y otros, 1991] es implementar los SBC usando ontologías (véase el capítulo 5) como soporte fundamental para las bases de conocimientos y métodos de resolución de problemas como motores de inferencia [Gómez y Benjamins, 1999]. Estos modos de representación y resolución se intentan construir de la manera más independiente posible al SBC que se va a desarrollar. De esta forma, pueden ser reutilizados para la construcción de otros sistemas con un menor esfuerzo. Por ejemplo, un método de resolución de problemas diseñado para planificación de forma general, puede ser utilizado para realizar un plan de construcción de un edificio o un plan de desarrollo de una aplicación software, seleccionando las ontologías necesarias e indicando los objetivos al método de resolución. De igual forma la ontología que fue usada para el plan de construcción de un edificio, puede ser utilizada por un SBC docente que enseñe procesos de construcción.

19.3.2 Propiedades de los SBCs

Como se ha comentado, los SBCs resuelven problemas no abordables por los sistemas de información tradicionales, pero para su construcción se requiere emplear un considerable esfuerzo debido, principalmente, al tiempo que conlleva la adquisición y conceptualización del conocimiento. Sin embargo, el desarrollo de SBCs está muy justificado por, entre otras, estas razones:

- Algunos problemas no tienen soluciones algorítmicas tratables en tiempo real. En cambio, los expertos humanos alcanzan un alto rendimiento debido a los conocimientos que tienen del medio y a la heurística empleada. Los SBCs tratan de igual forma de usar conocimientos y heurísticas para resolver problemas que no sería capaz de resolver la IS tradicional.
- Extraer el conocimiento de expertos cuesta mucho pero, una vez representado, se transforma en perdurable y además, puede ser reutilizado en varios sistemas.
- Permiten utilizar personal no especializado para resolver problemas. Un SBC bien construido permite servir de ayuda a personal en formación y, en algunas ocasiones, prescindir en gran medida de expertos con el consiguiente ahorro para las empresas.
- Las decisiones se realizan de una forma más eficiente. Esto no significa únicamente que las respuestas se generen con más rapidez sino que, además, las

soluciones serán más objetivas debido a que el SBC tendrá en cuenta las prioridades y probabilidades que se han indicado para tomar una decisión por muy crítica que sea y, en cualquier caso, pueden justificar las respuestas explicando la línea de razonamiento que hayan seguido.

Aunque estas ventajas son muy importantes, existen algunos inconvenientes que hacen que aún no se haya extendido de forma generalizada el empleo de SBC. El principal, como ya se ha apuntado, es el cuello de botella de la adquisición del conocimiento. Los expertos de un dominio no siempre están en disposición de perder su posición de privilegio con la competencia de un sistema que vaya a realizar su labor. Pero, aunque deseen colaborar, en la mayoría de las ocasiones es muy complicado obtener su forma de trabajo y generalizarla en un método. Pensemos que un experto, ante dos problemas similares, puede dar dos soluciones diferentes igualmente válidas. Además los expertos están en constante aprendizaje depurando su forma de razonar conforme van tratando nuevos casos. Estas características, aunque se persiguen en el desarrollo del SBC, son difíciles de implementar.

19.4 Métodos de desarrollo de sistemas basados en conocimiento

A finales de los años cincuenta comenzó el desarrollo de sistemas inteligentes, en principio aplicados en la confección de técnicas generales para la resolución de problemas. El más famoso fue el solucionador general de problemas de Newell y Simon [Newell y Simon, 1963] que introdujo las reglas de producción para imitar los métodos de resolución aplicados por los humanos. También en esa época se crearon otros sistemas como el sistema de planificación de STRIPS [Fikes y Nilsson, 1971], el programa de Samuel para el juego de las damas o el de Shannon para el ajedrez. Estos sistemas trataban de producir sistemas inteligentes que dependieran poco del dominio del conocimiento pero con procesos de razonamiento muy potentes.

A pesar de la euforia inicial en el campo de IA en la aplicación a diversos ámbitos, se descubrió que, cuando se quería utilizar una idea en otro dominio, se tenían que volver a hallar todos los principios básicos y, para problemas complejos, el funcionamiento nunca llegaba al nivel de un experto en la materia. La diferencia consistía en que un experto, además de tener una técnica para resolver problemas, disponía de una gran cantidad de conocimiento, de reglas que permitían inferir nuevo conocimiento e hipótesis y de experiencia en muchos casos prácticos que permitía generalizar los modos de actuación para dar una solución a un nuevo problema. La aceptación de que el conocimiento era la clave para abordar el desarrollo de sistemas aplicados a casos reales conllevó al auge de los SBCs.

Tomando como base el paradigma basado en conocimiento, a finales de los sesenta se crearon con éxito varios sistemas basados en conocimiento o sistemas expertos aplicados a diferentes dominios. Uno de los primeros fue DENDRAL [Buchanan y otros, 1969] que empleaba conocimiento heurístico extraído de expertos para poder interpretar espectogramas de masas con el fin de identificar componentes

químicos. Un paso muy importante para el desarrollo de los SBCs fue la construcción de MYCIN [Buchanan y Shortliffe, 1984] en el dominio del diagnóstico médico. Este sistema separó explícitamente la base de conocimiento del motor de inferencia, y significó que podría reutilizarse el núcleo esencial del sistema y sustituirse con otra base de conocimiento para otro dominio.

A continuación, se desarrollaron otros como XCON/R1 [Barker y O'Conner, 1989] para asistir a la configuración de pedidos de ordenadores o PROSPECTOR [Duda y otros, 1979] que obtuvo una gran fama debido a su empleo económicamente exitoso en la explotación de minerales.

Los anteriores sistemas se construían por el método de desarrollo “prueba y reparación de errores”. Fundamentalmente consistía en que el ingeniero preguntaba al experto cómo resolvía un conjunto de problemas, lo codificaba en forma de reglas y se probaba el prototipo, haciendo continuas versiones. Esta forma de desarrollo, aunque logró productos finales como los que se han comentado anteriormente, causaba enormes retrasos y costes, pérdida de fiabilidad y la incapacidad de abordar sistemas de cierta envergadura. Las metodologías de esta primera generación se centraban en la adquisición de los conocimientos y su inmediata incorporación al sistema en desarrollo. Entre estas metodologías se pueden citar Waterman [Waterman, 1986], Parsaye y Chignell [Parsaye y Chignell, 1988], Harmon y King [Harmon y King, 1985], y Carrizo [Carrizo y otros, 1989].

Para evitar los problemas ocasionados en la construcción de SBCs por este método de desarrollo incremental, en los años noventa la IC trató de aplicar los procesos y técnicas del modelo en cascada [Royce, 1970] empleados por la IS en la construcción de sistemas de información. Buchanan [Buchanan y otros, 1983] propuso esta adaptación indicando las fases de identificación de requisitos, conceptualización, formalización, implementación de reglas y validación, donde cada fase proporciona un producto a la siguiente, pudiéndose validar el trabajo en cada una de las fases y pudiendo refinar el sistema repitiendo estos procesos. Entre los métodos de desarrollo de SBCs usando el ciclo de vida en cascada se pueden mencionar los de Alberico y Micco [Alberico y Micco, 1990], POLITE [Edwards, 1991] o KADS [Wielinga y otros, 1992]. El problema principal de los métodos en cascada consiste en que la primera fase de identificación de requisitos debe estar completa y cerrada para continuar con el análisis y esto no suele darse en los SBC.

En los últimos años, inspirados en el modelo de Boehm [Boehm, 1988] para la construcción de sistemas de información, surgen las metodologías de desarrollo en espiral de SBCs. Estas metodologías proponen la construcción del sistema en ciclos, de forma que cada uno de ellos contendrá una definición de objetivos, una valoración de riesgos, un proceso de desarrollo y una planificación para abordar un nuevo ciclo de depuración. Este método es muy adecuado para ser aplicado a la IC, ya que es flexible a la modificación del sistema desarrollando un nuevo ciclo de procesos. Las metodologías más conocidas que han adaptado el método de Boehm a la IC son KLIC [Guida y Tasso, 1994], IDEAL [Gómez y otros, 1997], MIKE [Angele y otros, 1996] y CommonKADS [Schreiber y otros, 2000].

19.4.1 La metodología CommonKADS

La metodología CommonKADS (CK)¹ es la evolución de la metodología KADS (Knowledge Acquisition and Design Structuring) para el desarrollo de SBCs. Ha sido desarrollada por diversas universidades y empresas dentro de dos proyectos ESPRIT. Como indican sus siglas en inglés, su objetivo inicial era un método para la adquisición de conocimientos, pero actualmente se ha convertido en una de las más completas metodologías para el análisis, gestión y desarrollo de SBCs.

Como se estudiará en la siguiente sección, se basa en un conjunto de modelos que recogen todos los aspectos importantes que deben ser considerados para que un SBC tenga éxito. Los seis modelos que propone CK son: organización, tareas, agentes, conocimiento, comunicación y diseño [de Hoog y otros, 1994]. Estos están relacionados entre sí y se deben completar según las plantillas de documentos y técnicas descritas para cada uno. Para los modelos de conocimiento y comunicación, CK ofrece un lenguaje de representación propio llamado CML (Conceptual Modeling Language) [Anjewierden, 1997] que permite definir todos los elementos que van a intervenir en la parte conceptual del SBC permitiendo interrelacionarlos. En CML se propone la representación del conocimiento estático del dominio por un lado, y de las tareas e inferencias por otro, con el propósito de poder reutilizarlos en otros sistemas.

La actividad de modelado que sigue CK persigue el principio del nivel de conocimiento de Newell [Newell, 1982] que postula modelar el conocimiento a nivel conceptual separándolo del nivel de implementación en el sistema. De esta forma, el sistema se va construyendo desde los niveles más abstractos identificando objetivos y posibles problemas, pasando por el modelado conceptual del conocimiento que identifica los elementos estructurales, sus relaciones y elementos de comunicación, hasta llegar a aspectos prácticos de representación del conocimiento e implementación del sistema.

CK define un ciclo de vida en espiral compuesto por cuatro fases: 1) Una revisión (o estudio inicial si es el caso) del estado actual del proyecto y declaración de objetivos del sistema; 2) una identificación y valoración de riesgos para abordar las diferentes alternativas propuestas en la fase anterior; 3) una planificación de las tareas a desarrollar estableciendo los tiempos y recursos para cada trabajo; y 4) la monitorización del desarrollo del sistema para supervisar los tiempos, recursos empleados y calidades de los productos de cada tarea.

Se puede afirmar que CK constituye la metodología más completa, y la más utilizada en la actualidad, debido a que abarca todos los aspectos que deben ser tenidos en cuenta para el desarrollo con éxito de un SBC. Reúne técnicas de especificación de requisitos y análisis de objetivos propias de la IS adaptadas a la IC. Por otro lado, propone modelos de representación del conocimiento completos con el objetivo de favorecer la reutilización en diferentes sistemas y da las pautas para transformar estos modelos en un sistema implementado. Por estas razones, en la siguiente sección se describe esta metodología para que el lector estudie una forma de desarrollo completo de un SBC.

¹<http://www.commonkads.uva.nl>

19.5 Construcción de sistemas basados en conocimiento usando CommonKADS

En esta sección se describirán los modelos que establece la metodología CK para cada uno de los niveles de desarrollo de un SBC. Para ilustrar todo el progreso de desarrollo, se mostrará un caso de estudio relacionado con el control de refrigeración de una central nuclear. En cada modelo, se mostrará un breve ejemplo o unas indicaciones del contenido que debería incluir el InCo en cada uno de los documentos de trabajo. El lector debe tener en cuenta que los documentos de trabajo que se exponen aquí a modo de ejemplo sólo representan un pequeño esbozo de lo que supondría el documento completo del modelado.

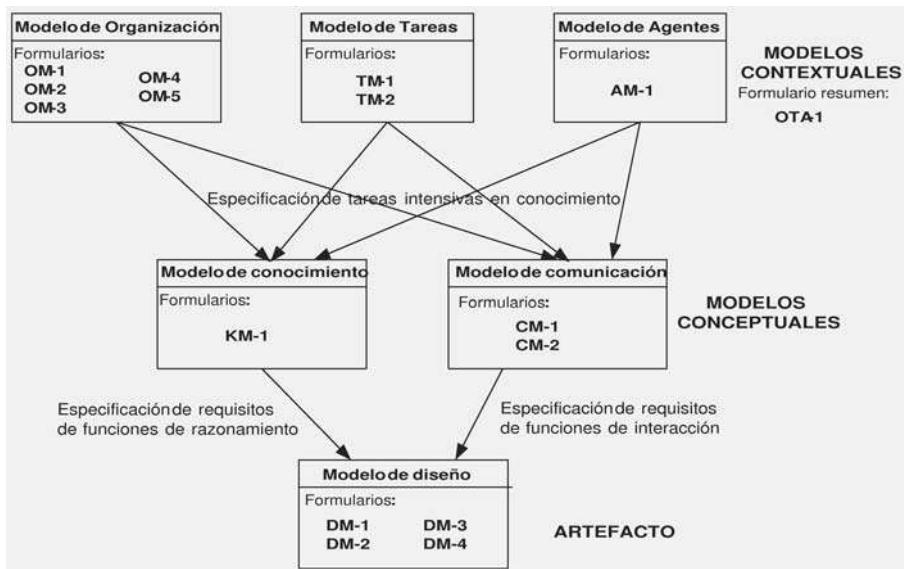


Figura 19.1: Esquema del conjunto de modelos de CommonKADS.

CK propone tres niveles en el desarrollo del sistema, que producirán un conjunto de modelos encargados de recoger los aspectos relevantes del nuevo sistema. En cada uno de los modelos, se especifican las plantillas o formularios que el InCo debe completar. Se presenta a continuación la estructura de los tres niveles de CK (cuyo esquema se muestra en la Figura 19.1) que serán descritos en las siguientes secciones con más detalle:

Nivel Contextual: se produce un informe detallado que analiza los objetivos, oportunidad, problemas, alternativas y elementos que intervienen con el propósito de decidir la adecuación de desarrollo de un SBC al dominio estudiado. El estudio contiene los siguientes elementos:

- **El modelo de la organización:** proporciona el análisis de las características principales de la organización con el objetivo de descubrir los problemas y oportunidades para realizar el SBC, indicar si es viable, y valorar el impacto de las acciones propuestas con el modelo. Contiene cinco formularios nombrados de OM-1 a OM-5.
- **El modelo de tareas:** recoge la disposición de las tareas principales que debe resolver el sistema, sus entradas y salidas, las precondiciones y los criterios de ejecución. Contiene los formularios TM-1 y TM-2.
- **El modelo de agentes:** los agentes son los ejecutores de las tareas. Este modelo describe las características de los agentes, en particular su competencia, autoridad para actuar y sus obligaciones y recursos. Contiene el formulario AM-1.
- Por último, el resumen final de los anteriores formularios se describe en un **informe de conclusiones** y acciones que deben llevarse a cabo especificadas en el formulario OTA-1.

Nivel Conceptual: describe la estructura y composición del conocimiento y los elementos de comunicación involucrados en las tareas. Contiene estos dos modelos:

- **El modelo de conocimiento:** especifica en detalle los tipos y estructuras del conocimiento usados para ejecutar una tarea. Proporciona una descripción independiente de la implementación sobre el papel que los diferentes componentes de conocimientos juegan en la resolución de un problema, y de forma que sea comprensible para los humanos. Esto hace posible la comunicación con expertos y usuarios sobre los aspectos de resolución de problemas de un SBC, durante el desarrollo y la fase de ejecución. El documento que recoge esta especificación es el KM-1.
- **El modelo de comunicación:** debido a que varios agentes y módulos pueden estar involucrados en la resolución de una tarea, es importante especificar las transacciones de comunicación entre ellos. También este modelo debe hacerse de manera conceptual e independiente de la forma en la que sea implementado. Los formularios de este modelo son CM-1 y CM-2.

Nivel Artefactual o de Implementación: se indica cómo pasar del nivel conceptual a la implementación del sistema, describiendo su arquitectura y sistema computacional. Consta del siguiente modelo:

- **El modelo de diseño:** Este modelo recogerá las especificaciones técnicas del sistema siguiendo los anteriores modelos. Describirá la arquitectura del sistema, las plataformas de ejecución, los módulos software, las bases de conocimiento necesarias, los métodos de resolución, los sistemas de comunicación, etc. Se utilizan los documentos DM-1 a DM-4.

19.5.1 Modelado del contexto en CommonKADS

De forma similar a las metodologías de IS, la primera fase que propone CK es la identificación del problema, análisis de requisitos a grandes rasgos, un estudio de adecuación del problema a ser tratado por un SBC y un planteamiento de acciones para desarrollar el sistema. En este nivel, cuya representación gráfica se muestra en la Figura 19.2, se realizan de forma secuencial dos estudios fruto de los modelos representados en los correspondientes formularios: un **estudio de viabilidad** tras la creación del Modelo de la Organización, y un **estudio de impacto y mejoras del sistema** tras la definición del Modelo de Tareas y del Modelo de Agentes usando unas técnicas específicas que se propondrán a continuación

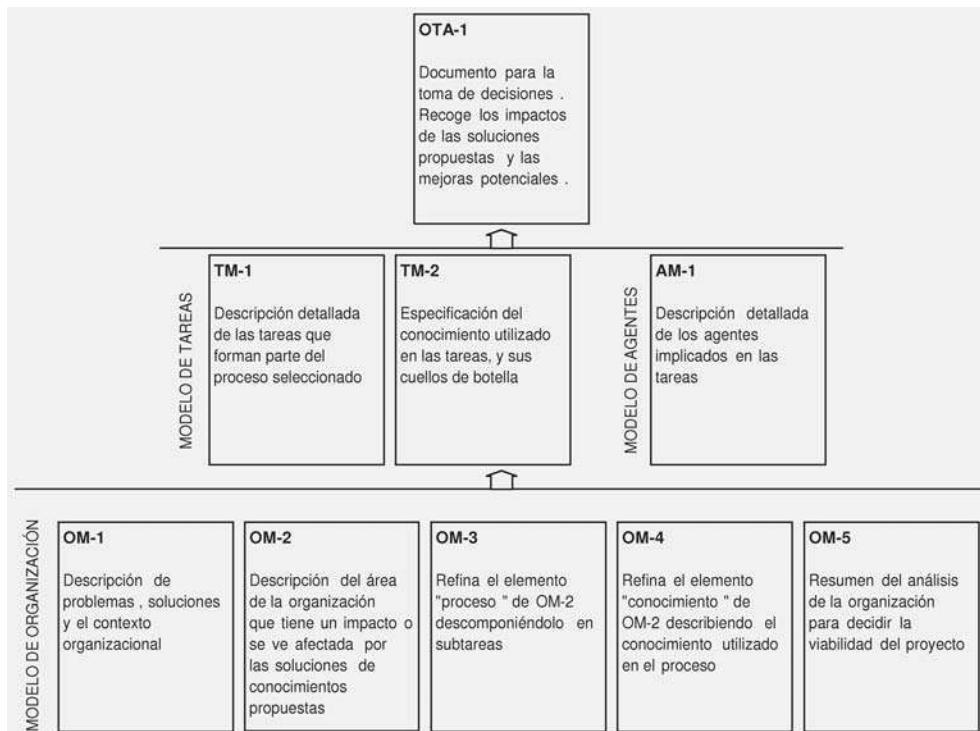


Figura 19.2: Esquema de la relación entre los formularios del nivel contextual.

En el estudio de viabilidad se identificarán las áreas de oportunidad o problemas y las soluciones potenciales, y se realiza una perspectiva amplia de la organización. Además, se analizan las características económicas, técnicas, y de viabilidad del proyecto para seleccionar el plan más favorable y proporcionar la mejor solución. En el estudio de impacto y mejora, por un lado se recopilan a grandes rasgos las relaciones entre las tareas y los agentes involucrados, y cómo se puede usar el conocimiento utilizado para la ejecución de las tareas de forma óptima, indicando las mejoras que pueden realizarse. Por otro, se aconsejan las medidas que deben llevarse a cabo en la

organización y los cambios en la realización de las tareas, para asegurar la aceptación e integración en la organización.

19.5.1.1 Análisis de viabilidad

Como se ha comentado, se describe la organización actual de una forma estructurada, los procesos, la plantilla y recursos indicando el papel que juega cada uno y cómo interactúan. Estos componentes se estudian con el fin de proporcionar nuevas soluciones. El análisis de viabilidad se corresponde con el modelo de la organización y comprende cinco apartados que se especifican en los correspondientes formularios o documentos. Como se indican en las Tablas 19.3 a 19.8 se van mostrando las descripciones, muy abreviadas en el ejemplo que se sigue, de la información que puede contener un caso de estudio de un sistema de control de la refrigeración de una central nuclear.

(OM-1) Documento de “Problemas y oportunidades”. Contiene una descripción a grandes rasgos de los objetivos, las estrategias, los valores y factores de influencia de la organización. Este conocimiento deben proporcionarlo los especialistas o expertos del área, los usuarios que van a manejar el conocimiento, y los directivos de la empresa que deben tomar decisiones con el conocimiento de la empresa. En los apartados de OM-1 se debe incluir la siguiente información:

- **Problemas y oportunidades:** es una lista reducida de los problemas y mejoras en la organización, basado en las entrevistas, reuniones, discusiones con directivos, etc.
- **Contexto organizacional:** Indica, de forma concisa, las características del contexto organizativo de la empresa, describiendo la misión, visión y metas de la organización, los factores externos importantes con los que debe tratar, la estrategia de la organización y los principales valores que deben mejorarse.
- **Soluciones:** se listan las posibles soluciones desde los problemas percibidos y cambio en las características de la empresa que deben realizarse.

(OM-2) Documento de “Descripción del área de interés de la organización -aspectos variables-”. Visualiza el modelo de la organización indicando cómo se estructuran los procesos de los negocios, la plantilla de personal que se necesita, los recursos, etc. El conocimiento se trata en un apartado como recurso especial por su importancia. La característica principal de estos componentes es que pueden cambiar como resultado de implantar el SBC. Los elementos de este documento son:

- **Estructura:** Disposición jerárquica de la empresa, incluyendo departamentos, grupos, secciones, etc.
- **Procesos:** Boceto de los procesos de negocios o forma de operar de la organización. Un proceso es una parte relevante de los valores que se intentan mejorar. Se descompondrán en tareas que se describirán en el formulario OM-3.

Modelo de Organización	Documento de problemas y oportunidades (OM-1)
PROBLEMAS Y OPORTUNIDADES	<ul style="list-style-type: none"> • El sistema software actual consiste en una serie de indicadores de sensores (de funcionamiento, temperatura, presión, nivel de llenado, etc.) situados en distintos elementos de la central nuclear. Se puede programar una serie de alarmas que avisen cuando los valores estén fuera de los rangos de seguridad. • Se echa en falta un sistema capaz de identificar la tendencia de los valores de las mediciones de los sensores y de aconsejar a los ingenieros los procedimientos adecuados para restablecer el funcionamiento normal del sistema de refrigeración. • El nuevo sistema debe ser probado sistemáticamente durante un largo periodo de tiempo. Permitirá dar respuestas más seguras y fiables y se podrá disminuir el número de ingenieros de control del sistema. • El SBC debe avisar de niveles fuera de rango, debe calcular las tendencias de los niveles, debe aconsejar las acciones que se han de llevar a cabo y sería conveniente que simulara el comportamiento del sistema de refrigeración para cada una de las acciones que se pudieran realizar. • Un valor añadido al implementar el SBC es la revisión sistemática de los procedimientos actuales de actuación ante contingencias.
CONTEXTO ORGANIZATIVO	<p>Misión, visión y meta: El sistema trata un proceso extremadamente crítico que no debería controlar sin intervención humana el sistema de refrigeración de la central. El objetivo principal es servir de apoyo a la supervisión del sistema y aconsejar las acciones que deben llevarse a cabo ante diferentes situaciones. Debe ser un sistema muy estudiado, consensuado y probado antes de su implantación. No se prevé que futuras versiones puedan controlar este proceso de forma automática sin intervención humana.</p> <p>Factores externos:</p> <ul style="list-style-type: none"> • El sistema basado en conocimiento ha sido encargado por una agencia estatal y no debe competir con otros productos ni mejorar los resultados comerciales de una empresa. • Existe una excelente disposición de los ingenieros de la central a que se desarrolle el sistema. Va a hacer que tengan más seguridad en sus acciones y permitirá que consideren todos los factores apuntados por el nuevo sistema. • El tiempo de vida de la central nuclear para la que se quiere desarrollar el sistema es de 9 años, quizás prorrogables. La estimación inicial es que el SBC se desarrolle e implante en 3 años.
SOLUCIONES	<p>Solución 1: Continuar con el actual sistema software consistente en dar avisos de alarma cuando los niveles de los sensores estén fuera de los parámetros correctos.</p> <p>Solución 2: Desarrollar el sistema de forma que controle automáticamente los elementos del sistema de refrigeración para restablecer los parámetros correctos. Esta solución es muy desaconsejada por el carácter crítico del sistema.</p> <p>Solución 3: El SBC se desarrolla con el objetivo básico de que aconseje las acciones que deben realizarse ante valores anormales de los sensores. Se pospone la simulación de los niveles del sistema tras efectuar las posibles acciones para posteriores versiones del SBC.</p> <p>Solución 4: El SBC debe desarrollarse para que indique a los ingenieros las acciones que se deben realizar y para que simule las consecuencias que acarreará en el sistema de refrigeración tras realizarlas.</p>

Tabla 19.3: Ejemplo de un formulario OM-1 de descripción de problemas y oportunidades de un sistema de control de refrigeración de una central nuclear.

- **Personal:** Se indican los miembros de la plantilla que se usan con el conocimiento de la empresa: los suministradores de conocimientos, los decisores, y los usuarios del conocimiento.
- **Recursos:** Describen los recursos utilizados en los procesos. Pueden ser de diferentes tipos, como sistemas de información u otros recursos software, equipamiento y materiales o tecnología de la empresa, patentes y derechos.
- **Conocimiento:** Representa un recurso especial explotado en los procesos. Debido a su importancia, se describe en un apartado por separado y será descrito con mayor detalle en el formulario OM-4.
- **Cultura y potencial:** Se presta atención a las reglas “no escritas”, los estilos de trabajo y comunicación, las relaciones interpersonales formales e informales, etc.

(OM-3) Documento de “Descomposición del proceso de negocio”. Los procesos se describen con mayor detalle, usando un formulario separado para cada uno de ellos. Los procesos usados en los negocios o en la operación de la organización se desglosan en pequeñas tareas, debido a que el SBC las ejecutará para completar el proceso completo. A menudo, se modificarán las tareas de los procesos existentes, o se combinarán de diferentes formas. Cada formulario OM-3 contendrá los siguientes campos:

- **Número:** Identificador que codifica la tarea para referenciarla en otros formularios.
- **Nombre de tarea:** Será una parte de un proceso de OM-2.
- **Realizada por:** Agente o sistema software identificado en OM-2 que se encarga de realizar la tarea.
- **Dónde:** Localización de la tarea en la estructura de la organización.
- **Recursos de conocimiento:** Lista de recursos de conocimientos usados por la tarea.
- **Intensiva en conocimiento:** Indica si la tarea usa de forma intensiva el conocimiento descrito.
- **Importancia:** Indicar cómo de relevante es la tarea, por ejemplo usando una escala fijada en términos de frecuencia, costes, recursos, si es o no crítica, etc.

Modelo de Organización	Documento de aspectos variables (OM-2)
ESTRUCTURA	El departamento objeto de estudio se denomina "Departamento de Refrigeración" y se engloba dentro del "Departamento de Producción Eléctrica". Están en contacto con el "Departamento de Mantenimiento y Reparaciones" para informarles para reparación del mal funcionamiento de los sensores y aparatos.
PROCESOS	Los procesos que llevan a cabo son la monitorización del sistema de refrigeración de la central. A través de circuitos de agua, disipan la energía calorífica en la producción eléctrica de la central nuclear. A niveles normales, los procesos de refrigeración son automáticos. El departamento realiza un chequeo periódico a lo largo del día de los sensores de todo el sistema. Deben realizar acciones críticas cuando ocurre algún imprevisto con mediciones fuera de los márgenes normales.
PERSONAL	Existe un Director, un Subdirector y 11 ingenieros en física para la supervisión del sistema de refrigeración y, al menos, 3 en cada momento en la central. Todos utilizan el sistema actual de monitorización de sensores.
RECURSOS	<ul style="list-style-type: none"> • Una sala amplia de operaciones compartida por todo el Dpto de Producción Eléctrica. • Un sistema de información que monitoriza los sensores del sistema de refrigeración y que regula las válvulas y demás elementos para mantener el sistema estable. • Cada ingeniero controla varios terminales (4 monitores cada uno). • El circuito de refrigeración y todas las características de los elementos que lo componen están documentados (en papel y formato electrónico). • Está documentado detalladamente en papel el procedimiento de actuación para todos los casos.
CONOCIMIENTO	El personal está muy altamente cualificado en física y especializado en energía nuclear. Conocen todos los elementos del sistema, su situación y características. Son capaces de tomar decisiones en situaciones críticas. Conocen perfectamente el manual de procedimiento.
CULTURA Y POTENCIAL	El trabajo es rutinario y el sistema funciona automáticamente en la mayoría de las ocasiones. Muy rara vez ocurre algún hecho en el que se deba actuar para variar algún indicador fuera de los límites de seguridad. Las incidencias más típicas consisten en mandar reparar algún sensor que no funciona adecuadamente. Pese a que el trabajo es monótono, el personal debe estar muy preparado para solucionar incidentes graves, ya que de ellos va a depender evitar una tremenda catástrofe. Los turnos de horario no son largos y, a menudo, se realizan simulacros para recordar el procedimiento de actuación.

Tabla 19.4: Ejemplo de formulario OM-2 de descripción de aspectos variables de la organización (2 de 2).

(OM-4) Documento de “Activos de conocimiento”. El conocimiento es el recurso más importante a analizar para el SBC, por ser el componente más valioso de la organización. Se usará un formulario para cada activo, y se detalla:

- **Recurso de conocimiento:** representa el nombre, ya identificado en OM-3, dado al conocimiento.
- **Pertenece a:** se identifican los agentes que lo poseen, y que deben aparecer en OM-3.
- **Usado en:** identifican las tareas que usan este conocimiento, también indicadas en OM-3

- **Forma, lugar, tiempo y calidad adecuados:** se indica si el conocimiento está siendo usado convenientemente en estas cuatro facetas y se comenta por qué se está usando así y cómo puede mejorarse.

Modelo de organización			Descomposición de los procesos (OM-3)			
Nº	TAREA	REALIZADA POR	¿DÓNDE?	CONOCIMIENTO	¿INTENSIVA?	IMPOR-TANCIA
1	Monitorización del sistema de refrigeración	Dpto. de Refrigeración	Sala de Operaciones	<ul style="list-style-type: none"> • Circuito y elementos • Uso del sistema de información 	sí	muy alta
2	Chequeo periódico de elementos del circuito	Dpto. de Refrigeración	Sala de Operaciones	<ul style="list-style-type: none"> • Circuito y elementos • Uso del sistema de información 	periódico, 8 veces al día	alta
3	Simulacro de incidentes	Dpto. de Producción (incluye Dpto de Refrigeración) y Dpto. de Seguridad	Sala de Operaciones	<ul style="list-style-type: none"> • Circuito y elementos • Uso del sistema de información • Procedimiento ante incidentes 	periódico, al menos 1 vez por semana	media
4	Actuación ante contingencias	Dpto. de Refrigeración	Sala de Operaciones	<ul style="list-style-type: none"> • Circuito y elementos • Uso del sistema de información • Procedimiento ante incidentes 	no	extrema

Tabla 19.5: Ejemplo de formulario OM-3 de descomposición de los procesos de la organización.

Modelo de la organización		Documento de Activos de conocimiento (OM-4)				
RECURSO DE CONOCIMIENTO	PERTENECE A	USADO EN	¿FORMA ADECUADA?	¿LUGAR ADECUADO?	¿TIEMPO ADECUADO?	¿CALIDAD ADECUADA?
Circuito y elementos	<ul style="list-style-type: none"> • Sistema de información • Dpto. de Refrigeración 	1, 2, 3 y 4	sí	sí	sí	sí
Uso del sistema de información	<ul style="list-style-type: none"> • Dpto. de Refrigeración 	1, 2, 3 y 4	sí	sí	sí	sí
Procedimiento ante incidentes	<ul style="list-style-type: none"> • Dpto. de Producción • Dpto de Seguridad 	3 y 4	sí, aunque podría respaldarse por un SBC	sí	sí	sí, aunque podría respaldarse por un SBC

Tabla 19.6: Ejemplo de formulario OM-4 de activos de conocimiento de la organización.

Documento de “Análisis de viabilidad” (OM-5). Con la información de los anteriores formularios, se realiza una valoración en conjunto de estos elementos en un documento, para que la directiva de la organización pueda tomar las decisiones oportunas. Este informe tiene gran importancia porque se debe justificar la inversión que se va a realizar en los cambios propuestos. Se compone de los siguientes elementos:

- **Viabilidad empresarial:** Para cada una de las áreas con problemas o mejorables, se describirán los beneficios tangibles e intangibles se esperan obtener, cuál es el valor añadido esperado, cuánto costará la solución, cuáles son las diferentes alternativas de solución, qué cambios organizativos deben llevarse a cabo, y qué riesgos e incertidumbres están vinculadas a la solución propuesta.
- **Viabilidad técnica:** Para cada una de las áreas con problemas o mejorables, se explicará cómo de complejo, en términos de representación de conocimiento y razonamiento están vinculados a la solución, cuál es la solución técnica que se propone, qué aspectos están involucrados respecto a los costes, tiempo, y calidad de la solución, qué riesgos e incertidumbres tecnológicos vinculados a la solución, etc.
- **Viabilidad del proyecto:** Para cada una de las áreas con problemas o mejorables, se responderán a cuestiones relacionadas directamente con las acciones que deben realizarse para llevar a buen fin el proyecto, por ejemplo si hay adecuado compromiso entre el personal involucrado en todos los pasos de ejecución del proyecto, si están disponibles los recursos y el conocimiento, la comunicación existente entre los elementos internos y externos de la organización, los riesgos e incertidumbres vinculados al proyecto, etc.
- **Acciones propuestas:** En este apartado se sopesa e integran los elementos anteriores y se recomiendan elementos concretos de actuación: en qué áreas se debe centrar el sistema, cuál es la solución final recomendada, qué resultados se esperan en costes y beneficios, qué acciones se requieren para llevar a cabo el proyecto, qué acciones tomar si cambian las condiciones de la organización, de la técnica o del proyecto, etc.

19.5.1.2 Análisis de Impacto y Mejoras

Si el estudio de viabilidad ha sido aprobado por la directiva de la empresa, se deberá ver en detalle las características de las tareas relevantes, los agentes que las llevarán a cabo, y el conocimiento que usarán para hacerlo. Se trata de refinar el modelo de la organización concretando los elementos y requisitos que necesitará el modelo conceptual del sistema. El análisis de impacto y mejoras comprende el modelo de tareas, el modelo de agentes y contiene un formulario resumen del nivel contextual.

(TM-1) Documento de “Análisis de tareas”. El concepto de **tarea** se refiere a una actividad que se realiza para alcanzar algún propósito, y a una subparte de un proceso de negocio. De esta forma, se caracteriza por representar una actividad

para mejorar algún valor de la organización, manejar entradas y transmitir salidas de forma controlada, consumir recursos, requerir (y suministrar) conocimientos, es realizada por un agente y debe seguirse algún criterio de calidad y rendimiento.

Modelo de la organización	Documento de análisis de viabilidad (OM-5)
VIABILIDAD EMPRESARIAL	Con el desarrollo de un SBC que respalde el plan de actuación ante contingencias se va a buscar como mayor objetivo ganar en seguridad sobre las acciones que deben realizarse ante algún incidente. Como efecto secundario, podrá reducirse el tiempo de formación de nuevos ingenieros y, si lo estima la empresa, reducir el número de ingenieros que monitorizan el sistema.
VIABILIDAD TECNICA	La inclusión en el SBC de la representación del conocimiento de los elementos del circuito y la identificación de las acciones que se deben realizar no conlleva ningún problema técnico. El SBC no necesitará de complejos mecanismos heurísticos ni idear nuevas formas de representación. Simplemente habrá que representar adecuadamente el conocimiento y los métodos de razonamiento de forma que indique las posibles acciones que deben realizarse, ya especificadas en los correspondientes manuales de procedimiento. Para que el SBC sea rentable, debe abordarse de forma inmediata ya que se ha estimado que el desarrollo del SBC será de 3 años y la caducidad de la central está en 9 años.
VIABILIDAD DEL PROYECTO	En este caso, la empresa y todo el personal del dpto. de refrigeración creen muy oportuno el desarrollo del SBC. No existe ningún problema en el presupuesto y la única dificultad es que no debe demorarse el comienzo del desarrollo.
ACCIONES PROPUESTAS	Se propone respaldar la toma de decisiones ante incidentes para ganar en seguridad y rapidez con el SBC. No se esperan beneficios empresariales tangibles. El SBC se desarrolla con la pretensión de que indique las acciones correctas de actuación ante el riesgo de una posible catástrofe nuclear.

Tabla 19.7: Ejemplo de formulario OM-5 sobre el análisis de viabilidad del nuevo sistema.

En el documento “análisis de tareas” se describen con más detalle las tareas que componen los procesos de la organización. Las características descritas en el documento TM-1 se refieren a aspectos tanto funcionales (entradas-salidas, flujos de datos, descomposición), sobre la estructura de la información que manejan, como del control que ejercen sobre otras y cuándo se realizan en el tiempo. Algunas de estas características ya se indicaron de forma más general en OM-3. Los elementos de este documento son:

- **Tarea:** Identificador y nombre de la tarea.
- **Organización:** Indica la parte del proceso de la que es parte, y en qué parte de la organización se realiza.
- **Objetivo y valor:** Indica los objetivos y los valores que añade al proceso.
- **Dependencia y flujo:** Se establecen las tareas precedentes y las tareas que dependen de ella.
- **Objetos manipulados:** Se identifican los elementos de información y conocimiento de los objetos de entrada, de salida e internos que se usarán para resolver la tarea.

- **Tiempo y control:** Describe la frecuencia, duración y control relacionada con otras tareas. Describe también las precondiciones y postcondiciones de la tarea.
- **Agentes:** Miembros de la plantilla o sistema de información responsable.
- **Conocimiento y capacidad:** Se debe indicar qué conocimiento (a grandes rasgos) y qué elementos son necesarios para que la tarea pueda conseguirse. Aquí también debe indicarse a qué elementos de la organización puede atribuirse la competencia.
- **Recursos:** Describe y cuantifica los recursos consumidos (tiempo de plantilla, sistema, equipos, material,).
- **Calidad y eficiencia:** Lista de las medidas de calidad y rendimiento que se usarán para determinar el éxito de realizar la tarea.

Modelo de tareas	Documento de análisis de tareas (TM-1)
TAREA	Chequeo periódico de elementos del circuito (Tarea 2 de OM-3).
ORGANIZACIÓN	Se realiza en el Dpto. de Refrigeración. Ante el fallo de algún elemento, se comunica a la Dirección de la Central y al Dpto. de Mantenimiento y Reparaciones.
OBJETIVO Y VALOR	El objetivo es hacer una comprobación exhaustiva de cada uno de los elementos del sistema de refrigeración. Es una tarea importante ya que los sensores, válvulas y demás elementos del circuito deben estar en perfecto estado para llevar a cabo su cometido.
DEPENDENCIA Y FLUJOS	No tiene dependencias con otra tareas, excepto que debe tenerse en cuenta la programación de la tarea 3 de simulacro de incidentes para que no coincida en el tiempo con esta tarea.
OBJETOS MANIPULADOS	Objetos de entrada: medidas de los elementos del circuito de refrigeración. Objetos de salida: identificación de los elementos del circuito con medidas anómalas.
TIEMPO Y CONTROL	Frecuencia: 8 veces al día. Duración: 22 minutos de media.
AGENTES	La realizan los 3 ingenieros del dpto de refrigeración que están en la sala de operaciones. El sistema de información actual va guiando el proceso de chequeo y advirtiendo si automáticamente se detecta algún valor anómalo.
CONOCIMIENTO Y CAPACIDAD	<ul style="list-style-type: none"> • Circuito de refrigeración y características de los elementos del circuito. • Uso del sistema software de información.
RECURSOS	Monitores del sistema de información.
CALIDAD Y EFICIENCIA	Se realiza con un protocolo muy seguro y sistemático. El sistema de información hace cumplir este protocolo obligando al ingeniero a validar la comprobación de los elementos del circuito.

Tabla 19.8: Ejemplo de uno de los formularios TM-1 sobre el análisis de tareas.

(TM-2) Documento de “Análisis de los cuellos de botella del conocimiento”. En este documento se describen los componentes de conocimiento (desde el documento OM-4) con más detalle, indicando las debilidades y cuellos de botella relacionadas con áreas específicas del conocimiento. El objetivo es mejorar el uso del conocimiento en la organización. La información sobre estas mejoras puede ser obtenida preguntando a

las personas directamente involucradas en su uso. Las características que se estudian se agrupan en tres áreas:

- **Naturaleza del conocimiento:** se describen las propiedades internas del conocimiento, si es fácil de adquirir, si maneja incertidumbre, si es un conocimiento basado en la experiencia o en la acción, etc.
- **Forma del conocimiento:** se indica en qué soportes está representado y se identifican los problemas para ser explotado.
- **Disponibilidad del conocimiento:** se especifican los obstáculos y limitaciones del conocimiento correspondiente a su estado.

Modelo de tareas	Documento de análisis de los cuellos de botella del conocimiento (TM-2)	
NOMBRE	Procedimiento ante incidentes	
POSEIDO POR	Dpto. de Producción (incluye Dpto. de Refrigeración) y Dpto. de Seguridad.	
USADO EN TAREA	Tareas 3 y 4	
DOMINIO	Incidente en el circuito de refrigeración	
Naturaleza del conocimiento	Sí/No	¿Cuello de botella? / ¿Qué se puede mejorar?
Formal, riguroso	Sí	
Empírico, cuantitativo	No	
Heurístico, sentido común	No	
Altamente especializado, específico del dominio	Sí	Sí/Podría respaldarse por un SBC
Basado en la experiencia	...	
Basado en la acción	No	
Incompleto	No	
Inciso, puede contener incorrecciones	...	La probabilidad de que se use mal este conocimiento es muy baja pero las consecuencias serían muy graves
Bastante cambiante	No	
Difícil de verificar	No	
Táctico, difícil de transferir	No	
Forma del conocimiento	Sí/No	¿Cuello de botella? / ¿Qué se puede mejorar?
Mental	Sí	
Papel	Sí	
Electrónico	No	Sí/Puede darse el caso de que se necesite en pocos segundos acceder al protocolo de actuación
Habilidades	No	
Otras	No	
Disponibilidad del conocimiento	Sí/No	¿Cuello de botella? / ¿Qué se puede mejorar?
Limitaciones de tiempo	Sí	Sí/Puede requerirse acceder al protocolo rápidamente
Limitaciones de espacio	No	
Limitaciones de acceso	No	
Limitaciones de calidad	No	
Limitaciones de forma	Sí	Sí/Debería estar transferido al SBC

Tabla 19.9: Ejemplo de uno de los formularios TM-2 sobre las características y cuellos de botella del conocimiento.

(AM-1) Documento de “Descripción de agentes”. El propósito es comprender los papeles y capacidades de los elementos activos de la organización (puede ser personal de la empresa o sistemas de información) para ejecutar tareas, normalmente compartidas. Aunque esta información puede estar ya especificada en otros documentos, puede servir para realizar cambios en las asignaciones de las tareas. Los apartados de este documento son:

- **Nombre:** nombre del agente.
- **Organización:** posición del agente en la estructura de la organización.
- **Implicado en:** indica las tareas (desde TM-1) en las que está involucrado.
- **Se comunica con:** otros agentes con los que intercambia información.
- **Conocimiento:** elementos de conocimiento (desde TM-2) que posee.
- **Otras competencias:** otras competencias del agente.
- **Responsabilidades y restricciones:** en la ejecución de las tareas.

Modelo de agentes	Documento de descripción de agentes(AM-1)
NOMBRE	Ingeniero de monitorización
ORGANIZACIÓN	Dpto. de Refrigeración
INVOLUCRADO EN	Tareas 1, 2, 3 y 4
COMUNICADO CON	Dpto. de Producción, Dirección de la Central, Dpto. de Seguridad, Dpto. de Mantenimiento y Reparaciones.
CONOCIMIENTO	<ul style="list-style-type: none">• Circuito de refrigeración y elementos• Uso del sistema de información• Procedimiento ante incidentes
OTRAS COMPETENCIAS
RESPONSABILIDADES Y OBLIGACIONES	Supervisar el estado del sistema de refrigeración. Comprobar los sensores del circuito. Llevar a cabo las acciones para restaurar valores anormales de funcionamiento del circuito. Estar preparado para aportar soluciones ante problemas críticos no previstos.

Tabla 19.10: Ejemplo de uno de los formularios AM-1 sobre la descripción de los agentes.

(OTA-1) Documento de “Recomendaciones y Acciones de mejoras”. Integra los documentos anteriores para servir a la toma de decisiones de cambios a la directiva de la organización. Aunque no se lleve a cabo la construcción del SBC, este estudio servirá para sacar a la luz muchas medidas y mejoras para la productividad de la empresa. En este documento se reflejan los siguientes aspectos:

- **Impactos y cambios en la organización:** describe qué implicará adoptar la solución indicada por la implantación del SBC respecto a la organización,

comparando las diferencias con el modelo de organización (OM-2) de la actual situación. Se hará para todos los aspectos variables, abordándose de una forma global.

- **Impactos y cambios en las tareas y agentes:** describe qué implicará adoptar la solución indicada por la implantación del sistema de conocimientos respecto a tareas y agentes individuales, comparando las diferencias con los modelos (TA-1, TA-2, AM-1) actuales. Se deben indicar:
 - Cambios en la disposición de las tareas (dependencias, objetos, tiempo,...).
 - Cambios en los recursos necesarios.
 - Criterios de ejecución y calidad.
 - Cambios en la plantilla, jerarquía, responsabilidades,...
 - Cambios en los conocimientos requeridos y capacidades.
 - Cambios en la comunicación.
- **Actitudes y compromisos:** se considera cómo pueden reaccionar los actores involucrados en los cambios propuestos.
- **Acciones propuestas:** directamente relacionado con los compromisos de dirección que decidirán los cambios. Se sopesarán e integrarán los análisis anteriores, recomendando pasos concretos de actuación sobre mejoras concretas para los cambios recomendados y cómo medir estos cambios, qué resultados se esperan tras la implantación y qué acciones realizar si cambian las circunstancias que rodean a la organización.

19.5.2 Modelado conceptual en CommonKADS

En este apartado se especifica la conceptualización del sistema en CK, basada en el Modelo de la Organización del apartado anterior y que servirá como entrada para la construcción del Modelo de Diseño de la etapa posterior. Abarca la realización del Modelo de Conocimiento que representa los conocimientos y requerimientos de razonamiento para el futuro sistema, y el Modelo de Comunicación que especifica las necesidades del sistema respecto a la interacción entre los agentes internos o externos.

19.5.2.1 El Modelo de conocimiento

Aquí se describen en detalle las estructuras y elementos del conocimiento del SBC, sin hacer referencia a los detalles de implementación. Para construir este modelo, CK se basa en la notación de UML [Booch y otros, 1997] como esquemas gráficos de representación de los distintos componentes, y un lenguaje propio semiformal denominado Lenguaje de Modelado Conceptual (CML) [Anjewierden, 1997]. De igual forma que en la sección anterior, a continuación sólo se muestra en cada apartado un ejemplo muy simple del contenido de estas representaciones para que el lector pueda intuir el contenido que debe especificarse. Se advierte que no se describe el formato de los

elementos en UML ni tampoco la sintaxis de CML. Si se quiere aprender estas notaciones, se debe consultar [Booch y otros, 1997] y [Anjewierden, 1997] y si se desean estudiar ejemplos más completos se recomienda leer [Schreiber y otros, 2000], [Alonso y otros, 2004], [Pajares y Santos, 2005], [Linster y Musen, 1992], [Sacile, 1995] o [Taboada y otros, 1999].

Modelos de organización, tareas y agentes	Documento de "Recomendaciones y acciones de mejoras" (OTA-1)
IMPACTOS Y CAMBIOS EN LA ORGANIZACIÓN	La implantación del SBC no va a suponer una modificación en la organización de la empresa. Únicamente debería incorporarse un Ingeniero del Conocimiento para supervisar el funcionamiento del sistema, incorporar nuevos elementos que se modifiquen en el sistema de refrigeración y recoger los cambios de procedimiento que se hagan en las medidas de chequeo y acciones ante incidentes.
IMPACTOS Y CAMBIOS EN LAS TAREAS Y AGENTES	En el trabajo rutinario, no se debe realizar ninguna modificación. Sólo en el caso de producirse algún incidente, los ingenieros deberán consultar el SBC para verificar el plan de actuación en la resolución del problema.
ACTITUDES Y COMPROMISOS	La directiva de la empresa y el personal del dpto. de refrigeración acogen el desarrollo del SBC con muy buena disposición. Va a respaldar las decisiones que tomen y va a dar más rapidez ante incidentes graves. El desarrollo de un SBC debe hacerse con precisión y detalle pero, en este caso, con mayor motivo. Se estudiarán todos los posibles casos y se revisará el protocolo de actuación.
ACCIONES PROPUESTAS	Se decide realizar cuanto antes el desarrollo del SBC, optando por la solución 3 indicada en OM-1. Esto es, implementar un SBC que indique las acciones que deben llevarse a cabo cuando algunos de los indicadores tengan valores anómalos. En definitiva, es implementar en el SBC los manuales de actuación existentes en la central nuclear. No se aborda, por el momento, el módulo de simulación en los indicadores del sistema tras probar un conjunto de acciones. Se pospone debido a la falta de tiempo en tener el sistema útil antes de que se cierre la central nuclear. Las acciones que deben llevarse a cabo son, en primer lugar, catalogar y representar los elementos del sistema de refrigeración (sensores de presión, temperatura, válvulas, compuertas, niveles de depósitos, etc.) que actualmente están en una base de datos del sistema de información. Se representarían en el SBC con los correspondientes atributos y relaciones entre ellos. En segundo lugar, se representarían las reglas de actuación que deben conectar los valores de los indicadores con los elementos que pueden modificar estos valores (válvulas de presión, compuertas, llaves de paso, etc.). Por último, se representarán los métodos que van a determinar qué reglas deben emplearse según el estado de los indicadores del sistema de refrigeración.

Tabla 19.11: Ejemplo de formulario OTA-1 sobre recomendaciones y acciones de mejora en la organización (2 de 2).

El Modelo de Conocimiento posee tres categorías, que se describen con más detalle en los apartados siguientes:

- **Conocimiento del dominio:** identifica el conocimiento específico del dominio y los tipos de información sobre la que trata el sistema. Por ejemplo, se puede definir qué es una válvula de conducción doble o un sensor de presión

con todos sus atributos y relaciones jerárquicas, o también representar el hecho de que “sensor st744 tiene una temperatura de 134°C”, e incluir reglas del tipo “Si (sensor_tipo_sht tiene temperatura mayor que 93°C y conducto_helio_relacionado_con_sensor_tipo_sht está abierto) Entonces Abrir (válvula_secundaria_tipo_vhp_relacionada_con_sensor_tipo_sht) y Revisar (válvula_primeria_tipo_vhp_relacionada_con_sensor_tipo_sht)”.

- **Conocimiento sobre inferencias:** describe los pasos de razonamiento básicos que se deben usar en un sistema. Son los elementos que usarán los motores de inferencia para hacer deducciones con el conocimiento del dominio. Se relacionan de esta forma los elementos descritos en el conocimiento del dominio, por ejemplo cómo se vinculan válvulas, conductos, sensores, hechos del estado actual del circuito, y las reglas que relacionan estos elementos.
- **Conocimiento sobre tareas:** representa las metas que persigue el sistema, y cómo pueden descomponerse estas metas en subtareas y procesos de razonamiento, indicando mecanismos de control interno. Se describirán aquí cómo se utiliza el conocimiento del dominio y el conocimiento de razonamiento para obtener una salida. Por ejemplo, para realizar una acción o una petición de revisión del circuito de refrigeración, qué hechos e inferencias deben escogerse en cada ciclo de deducción y qué pasos deben seguirse conforme se van obteniendo resultados.

Además de UML y CML para documentar el conocimiento de estas tres categorías, CK propone utilizar esquemas ilustrativos para apoyar esta representación y, además, un formulario denominado KM-1 en el que se recogerá toda la información útil que se ha ido adquiriendo en el modelado. En la Tabla 19.12 se muestra un breve ejemplo, haciendo referencia a supuestos documentos adjuntos que contendrían la información correspondiente. El significado de los apartados es el siguiente:

- **Modelo de conocimiento:** aquí se especifica de forma completa el modelo de conocimiento en texto acompañado de esquemas ilustrativos.
- **Fuentes de conocimiento usadas:** se indican las fuentes de conocimiento que se han empleado en la adquisición del conocimiento.
- **Glosario:** se listan los términos empleados y su definición.
- **Componentes considerados:** se identifican los componentes que se han estudiado para reutilizarlos en el sistema.
- **Escenarios:** se describen casos reales de resolución de diferentes problemas del dominio.
- **Resultados de validación:** se describen los resultados de la simulación del sistema frente a casos de prueba.

- **Material de adquisición de conocimiento:** se identifica el material (documentación de entrevistas, grabaciones, etc.) recopilado en la adquisición de conocimiento.

Modelo de Conocimiento	Documento sobre el Modelo de Conocimiento (KM-1)
MODELO DE CONOCIMIENTO	El SBC contiene la descripción de todos los elementos del sistema de refrigeración de la central nuclear y sus relaciones. Se representan, además, el conjunto de acciones que pueden realizarse sobre algunos de ellos y la manera de seleccionar las acciones que se deben realizar conforme el estado de los componentes del circuito. El detalle de todos estos elementos se describe en el documento adjunto <i>SBC_Refr_MC.pdf</i> .
FUENTES DE CONOCIMIENTO USADAS	Las fuentes principales han sido los planos de la central, el libro de especificación de componentes, los manuales de procedimiento ante incidentes existentes y el conocimiento de los ingenieros del Dpto. de Refrigeración.
GLOSARIO	En el documento adjunto <i>SBC_Refr_GT.pdf</i> se describen todos los términos usados en el sistema.
COMPONENTES	No se han encontrado bases de conocimiento ni ontologías que hayan podido ser reutilizadas en el sistema.
ESCENARIOS CONSIDERADOS	Se han recogido 83 partes de incidencias ocurridas en los últimos 24 meses. Las copias de estos partes se encuentran en la carpeta adjunta <i>SBC_Refr_Partes</i> .
RESULTADOS DE VALIDACIÓN	Se han simulado con éxito 750 procesos de incidencias, incluyendo los 83 reales. Los resultados se especifican en el documento adjunto <i>SBC_Refr_Test.pdf</i> .
MATERIAL DE ADQUISICIÓN DE CONOCIMIENTO	La documentación obtenida en el proceso de adquisición se encuentra en la carpeta adjunta <i>SBC_Refr_Adq</i> .

Tabla 19.12: Ejemplo de formulario KM-1 sobre la documentación del Modelo de Conocimiento.

El conocimiento del Dominio. Describe el conocimiento básico y relevante del dominio sobre el que se va a desarrollar el SBC. En esta representación se hace referencia explícita a los conocimientos propios del dominio por lo que, a diferencia del conocimiento sobre inferencias y sobre tareas, este conocimiento sólo será reutilizable en sistemas que traten sobre el dominio del SBC. Contiene dos tipos de elementos: esquemas del dominio y bases de conocimiento. Usando la notación de CML, la estructura del conocimiento del dominio es la que se muestra en la Figura 19.3. Se debe resaltar que un SBC puede contener más de un esquema del dominio y más de una base de conocimiento porque así se decida para organizar los distintos elementos que componen el sistema.

```
DOMAIN-KNOWLEDGE <nombre del SBC>
    <Descripción de los esquemas del dominio>
    <Descripción de las bases de conocimiento>
END DOMAIN-KNOWLEDGE<nombre del SBC>
```

Figura 19.3: Esqueleto del conocimiento del dominio en CML.

- **Conocimiento del Dominio: El esquema del dominio.**

Es una descripción esquemática del conocimiento e información estática específicos del dominio a través de un número de definiciones de tipos. Comparando con las metodologías de IS, se asemeja al modelo de datos. Usando la notación de CML, la estructura del esquema es la que se muestra en la Figura 19.4.

```
DOMAIN-SCHEMA <nombre del esquema>
  USES: <otros esquemas>
    <definiciones de tipos>
  END DOMAIN-SCHEMA <nombre del esquema>
```

Figura 19.4: Esqueleto del esquema de datos en CML.

En la cláusula USES, se especifican otros esquemas que podrían ser reutilizados por el SBC que se está desarrollando. A continuación, se especificarán las definiciones de tipos que son: conceptos, relaciones y tipos de reglas.

Un **concepto** o clase (CONCEPT en CML) describe un conjunto de instancias u objetos que aparecen en el dominio de una aplicación y que poseen las mismas características. La forma de describir las características de los conceptos es mediante la definición de atributos (ATTRIBUTES en CML). A estos atributos, se le asignan valores que se pueden completar en las instancias o en los propios conceptos. Se necesita indicar el tipo de valor que puede contener cada atributo (numérico, fecha, conjunto de valores, universal, etc.) o se puede definir uno particular (con VALUE-TYPE en CML). También se puede especificar la cardinalidad (CARDINALITY en CML) para indicar el número de valores que puede contener un atributo. Mediante axiomas (AXIOMS en CML) se pueden establecer restricciones sobre los valores de los atributos. Para formar la jerarquía de conceptos, se debe indicar que un concepto es subtipo de otros o que uno es supertipo de otros más específicos (SUB-TYPE-OF y SUPER-TYPE-OF respectivamente, en CML). También se pueden modelar asociaciones entre conceptos del tipo “parte de” (PART-OF y HAS-PARTS, en CML). En la Figura 19.5 se muestran algunos de estos elementos.

Aunque no es fácil saber dónde está el límite, una razón importante para definir algo como un concepto separado y no como un atributo de otro concepto es que tenga entidad suficiente sobre su propia existencia independiente de otros conceptos. Se debe señalar también que la especificación de los conceptos debería intentar hacerse con independencia de la tarea que deben resolver para que sean reutilizables, pero en la práctica esto resulta muy difícil debido a la gran cantidad de conocimiento que se debería incluir en cada concepto.

A parte de las relaciones de especialización y generalización para formar la estructura jerárquica entre conceptos (SUB-TYPE-OF y SUPER-TYPE-OF en CML), se pueden definir relaciones **ad-hoc** (BINARY-RELATION y RELATION en CML) entre dos o más conceptos definidos en el esquema. A través de sus argumentos, se asocian los conceptos y especificando la cardinalidad se restringe el número de conceptos que pueden intervenir en las relaciones.

```

Depósito_de_Helio_HG3
Situac_Llenado: REAL;
Presión: REAL;
Estado: estado_depósito;
Fecha_Caduc: DATE;
Ubicación_en_circuito: STRING;
...

CONCEPT Depósito_de_Helio_HG3;
DESCRIPTION: "Es un tanque metálico con gas helio de 200 l y 300 kg usado para enfriar rápidamente zonas críticas.
No se usa si no ocurre ningún incidente";
SUB-TYPE-OF: Depósito_de_Helio;
ATTRIBUTES:
Situac_Llenado: REAL;
Presión: REAL;
Estado: estado_depósito;
Fecha_Caduc: DATE;
Ubicación_en_circuito: STRING;
...
AXIOMS
0 <= Situac_Llenado <= 200
END CONCEPT Depósito_de_Helio_HG3;

VALUE-TYPE estado_depósito;
VALUE-LIST: {operativo, estropeado, almacén, reserva, vacío, caducado};
TYPE: ORDINAL;
END VALUE-TYPE estado_depósito;

Válvula Depósito Helio HG3
Estado: {cerrada, abierta}
Situac_Válvula: {operativa, estropeada, almacén}
Ubicación_en_circuito: STRING;
...

CONCEPT Válvula_Dep_de_Helio_HG3;
DESCRIPTION: "Válvula de apertura y cierre unida a un depósito de helio HG3. Por seguridad, puede haber varias
válvulas conectadas en paralelo para abrir el depósito de helio";
ATTRIBUTES:
Estado: {cerrada, abierta}
Situac_Válvula: {operativa, estropeada, almacén}
Ubicación_en_circuito: STRING;
...
END CONCEPT Válvula_Dep_de_Helio_HG3;

```

Figura 19.5: Ejemplo de definición de conceptos en CML.

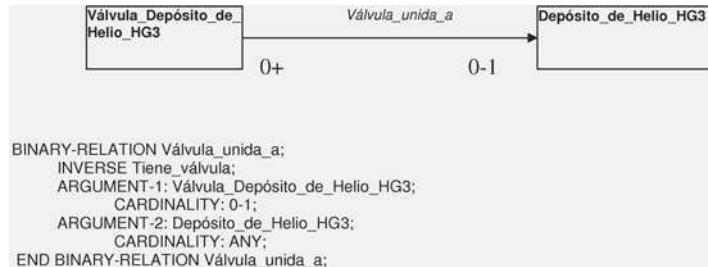


Figura 19.6: Ejemplo de relación binaria en CML.

Se pueden, además, asociar atributos a las relaciones. En la Figura 19.6 se muestra una relación binaria entre dos elementos.

Por último, en el esquema del dominio se permite definir un elemento característico de CK, que son los tipos de reglas. Los **tipos de reglas** son los esquemas o plantillas que se usarán para representar las reglas que formarán la base de conocimiento. Una regla se puede definir como una relación entre dos afirmaciones lógicas. Estas afirmaciones no implican dependencias estrictas, sino que pueden ser heurísticas u opiniones de expertos. Las afirmaciones lógicas se relacionan con expresiones sobre el valor de un atributo de un concepto.

Siguiendo el ejemplo de la central nuclear, supongamos que un tipo de regla que podemos usar es la que se corresponden con “sugerir una acción cuando la presión en un conducto es excesiva”. Así, se puede crear una plantilla de reglas que se usarán en las bases de conocimiento haciendo referencia a componentes concretos del circuito de refrigeración. Una posible definición de este tipo de reglas se muestra en la Figura 19.7. En ella se indica que cuando se definen las reglas correspondientes, en el antecedente debe aparecer algún sensor de presión (supongamos que es una clase bastante general y habrá bastantes subtipos de esta) y en el consecuente debe aparecer algún elemento de descompresión (supongamos que pueden ser válvulas, compuertas, grifos electroestáticos, bombas de fuga de seguridad, elementos de enfriamiento, etc.). El elemento de conexión que se usará en las reglas será Descompresiona_con. De esta forma, se permite modelar un conjunto de reglas que comparten una estructura y finalidad similar.

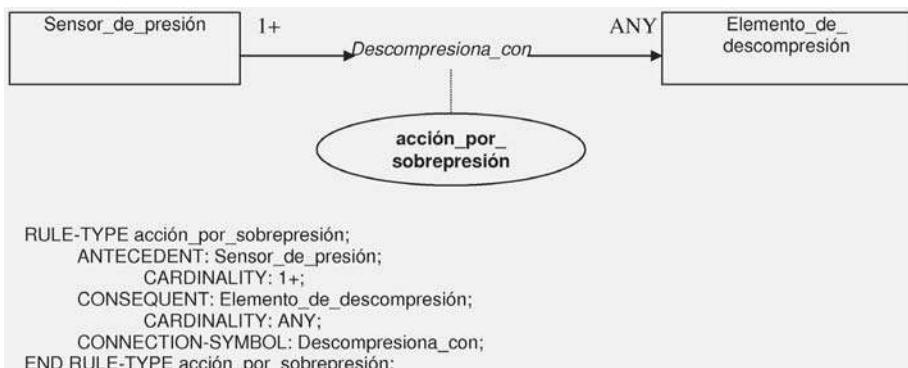


Figura 19.7: Ejemplo de tipo de regla en CML.

- **Conocimiento del Dominio: La base de conocimiento**

En la base de conocimiento se definen las instancias de los tipos especificados en los esquemas del dominio. Es decir, aquí se definirán instancias de conceptos, instancias de relaciones entre instancias de conceptos (mediante las correspondientes notaciones de INSTANCE en CML) y reglas (haciendo referencia a CONNECTION-SYMBOL) usando las plantillas de los tipos de reglas definidos

en los esquemas del dominio. Es usual disponer de varias bases de conocimientos con diferentes tipos de conocimientos.

Debe indicarse que, una vez especificado el esquema del dominio, lo usual es que éste se modifique muy rara vez, únicamente para incorporar los elementos que hayan cambiado en el dominio de la aplicación del sistema. En cambio, las bases de conocimiento son más modificables ya que, por un lado, las instancias de conceptos y relaciones son objetos cambiantes que se pueden incorporar y eliminar del SBC. Por otro, las reglas son conexiones entre elementos que pueden modificarse en el transcurso de operación del SBC por adquirirse nuevo conocimiento o por encontrarse con casos de estudio no contemplados.

Usando la notación de CML, la estructura de las bases de conocimiento es la que se muestra en la Figura 19.8. Tras la cláusula USES, se pueden importar todos los elementos de un esquema del dominio o sólo algunos indicándolo con la notación FROM. Después, tras la cláusula EXPRESSIONS, se declararán las instancias de los correspondientes conceptos, relaciones y reglas. En la Figura 19.9, se muestra un breve ejemplo de parte de una base de conocimiento que usa algunos de los elementos definidos en el esquema del dominio.

```
KNOWLEDGE-BASE <nombre de la base de conocimiento>
USES:
    <esquemas del dominio>;
    <elementos del esquema> FROM <esquema del dominio>;
EXPRESSIONS:
    <instancias de la base de conocimiento>
END KNOWLEDGE-BASE <nombre de la base de conocimiento>
```

Figura 19.8: Esqueleto de las bases de conocimiento en CML.

El Conocimiento sobre Inferencias Como se ha mostrado en el apartado anterior, el conocimiento del dominio describe las estructuras de conocimiento e información estáticas de un dominio. El conocimiento sobre inferencias describe cómo estas estructuras estáticas se enlazan para llevar a cabo un proceso de razonamiento. Los principales ingredientes son las inferencias, los roles de conocimientos y las funciones de transferencia, y se definen en la notación CML, como se muestra en la Figura 19.10.

- **Conocimiento sobre Inferencias: Las inferencias**

Describen el nivel inferior de la descomposición funcional. Son las unidades básicas del procesamiento de la información en el modelado del conocimiento. Una inferencia (INFERENCE en CML) usa el conocimiento contenido en alguna base de conocimiento para derivar nueva información desde su entrada dinámica. Las inferencias son tomadas como funciones primitivas que se definen completamente mediante la especificación declarativa de sus entradas y salidas. El proceso interno implementado en la inferencia es visto como una caja negra, y no se considera de interés para el modelado del conocimiento. La declaración de las inferencias se realiza de la manera más genérica posible con el objetivo de su reutilización en otros SBCs.

```

KNOWLEDGE BASE BC_central_nuclear1;
USES:
    esquema_central_nuclear;

EXPRESSIONS:

/* Instancia de concepto */
INSTANCE dh_117;
    Instance-of: Depósito_de_Helio_HG3;
ATTRIBUTES:
    Situac_Llenado: 132;
    Presión: 3;
    Estado: operativo;
    Fecha_Caduc: 05/10/2012;
    Ubicación_en_circuito: circuito 117B;
    ...
END INSTANCE dh_117;

/* Instancia de relación */
INSTANCE rel_val117;
    Instance-of: Válvula_unida_a;
    ARGUMENT-1: val34;
    ARGUMENT-1: val35;
    ARGUMENT-1: val36;
    ARGUMENT-2: dh117;
END INSTANCE rel_val117;

/* Instancias de reglas*/
sensor_presP4.presión > 3   Descompresiona_con compuerta_presa.estado = abierto
sensor_depHG3.presión > 6   Descompresiona_con válvula_Dep_de_Helio_HG3.estado = abierta;
bomba_fuga.estado = activa  Descompresiona_con válvula_grifoG84.apertura >= 73;

END KNOWLEDGE-BASE BC_central_nuclear1;

```

Figura 19.9: Ejemplo de parte de una base de conocimiento en CML.

```

INFERENCE-KNOWLEDGE <identificador del conocimiento de inferencia>
    <Descripción de inferencias>
    <Descripción de las funciones de transferencia>
    <Descripción de los roles de conocimiento>
END INFERENCE-KNOWLEDGE<identificador del conocimiento de inferencia>

```

Figura 19.10: Esqueleto del conocimiento de inferencia en CML.

- **Conocimiento sobre Inferencias: Roles de conocimiento**

Las entradas y salidas de las inferencias se describen en términos de roles de conocimiento (KNOWLEDGE-ROLE en CML) funcionales: nombres abstractos de objetos de datos que indican su papel en el proceso de razonamiento. Aquí es donde las inferencias se vinculan directamente con el SBC que va a manejarlas, asociando cada rol funcional con elementos concretos de la base de conocimiento.

Se distinguen dos tipos de roles de conocimientos: dinámicos y estáticos. Los dinámicos son las entradas y salidas de las inferencias en tiempo de ejecución. Cada invocación de la inferencia suele tener diferentes instanciaciones de los roles dinámicos. Por otro lado, los roles estáticos son más estables en el tiempo;

estos especifican el papel estático del conocimiento del dominio que se usa para hacer la inferencia.

Para ilustrar un ejemplo con las definiciones de inferencia y rol de conocimiento, en la Figura 19.11 se muestra la declaración de la inferencia descomprime. Se puede observar cómo su declaración es muy abstracta y puede usarse para el SBC de la central nuclear pero también para otros dominios como en un SBC de reparación de calefacciones o, divagando mucho, para un sistema médico antiestrés. Los términos son muy generales y sólo cuando en el rol se enlazan cada uno de los elementos con la especificación de la central nuclear es cuando se especializa su función.

En la figura se puede observar gráficamente cómo se realiza una proyección en el dominio de la inferencia, de forma que los roles de una inferencia se emparejan con los tipos definidos en el conocimiento del dominio. Con los roles del conocimiento se posibilita la construcción de catálogos de patrones de razonamiento reutilizables. El precio que se tiene que pagar es el incremento de la complejidad en la definición de las funciones.

• **Conocimiento sobre Inferencias: Funciones de transferencia**

En ellas se definen las funciones de comunicación con otros agentes externos (usuarios o sistemas). Estas funciones son cajas negras que indican cómo se producen las entradas y salidas desde el sistema con otros agentes. De igual forma que para las inferencias, se necesitará especificar los roles de las entradas y de las salidas. Existen cuatro tipos especificados en CK:

- OBTAIN (obtener): El agente de razonamiento requiere una parte de información del agente externo. El sistema tiene la iniciativa.
- RECEIVE (recibir): El agente de razonamiento da una parte de información al agente externo. El agente externo tiene la iniciativa.
- PRESENT (presentar): El agente de razonamiento muestra una parte de información al agente externo. El sistema tiene la iniciativa.
- PROVIDE (proporcionar): El agente de razonamiento proporciona una parte de información al agente externo. El agente externo tiene la iniciativa.

En el ejemplo de la Figura 19.12 se declara una función de transferencia del tipo OBTAIN. Desde el conocimiento de la situación esperada, se busca información del agente externo, que la ratificará o no. La salida será la situación_actual.

El Conocimiento sobre Tareas El conocimiento de la tarea es la categoría del conocimiento que describe las metas y estrategias que deben seguirse para alcanzar los objetivos para los que se quiere diseñar el SBC. Se suele representar e indicar de forma jerárquica: desde la tarea más genérica se descompone en subtareas más específicas, hasta enlazar con las inferencias y funciones de transferencia. Existen dos tipos de conocimientos para especificar el conocimiento sobre tareas, que se definen en CML, como se muestra en la Figura 19.13:

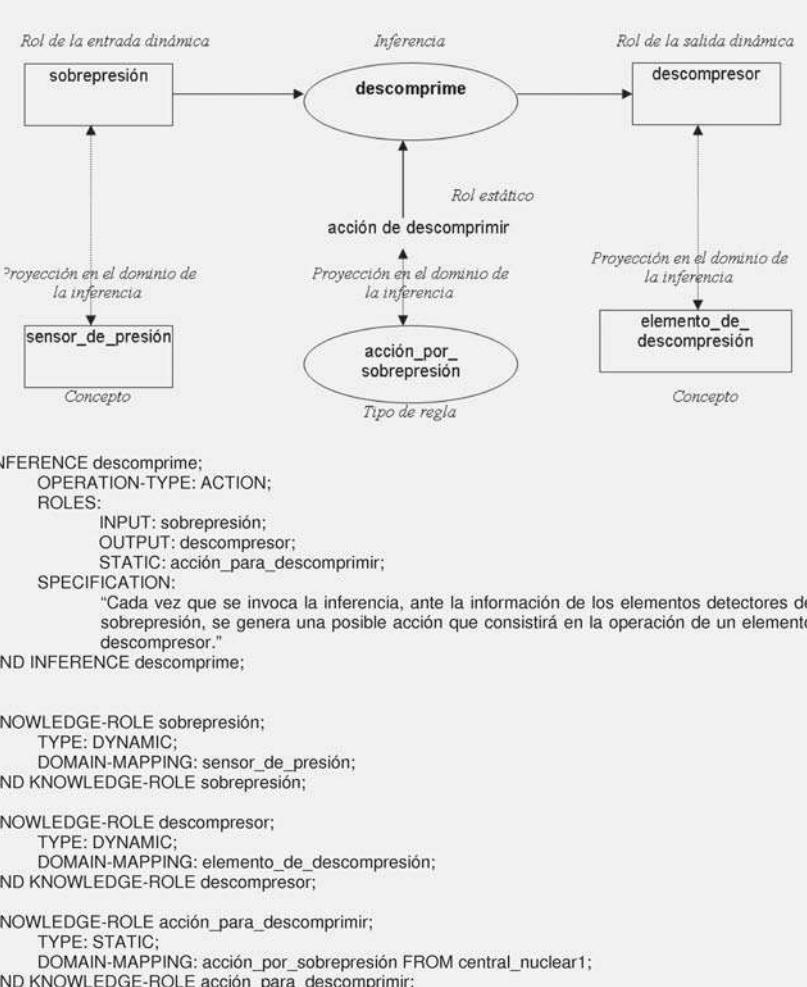


Figura 19.11: Ejemplo de inferencia y roles de conocimiento en CML.

```

TRANSFER-FUNCTION monitorización_sensores;
  TYPE: OBTAIN;
  ROLES:
    INPUT: situación Esperada;
    OUTPUT: situación Actual;
  END TRANSFER-FUNCTION monitorización_sensores;

```

Figura 19.12: Ejemplo de función de transferencia en CML.

- La **tarea** define la meta del proceso de razonamiento en términos generales, en función de sus entradas y salidas. En definitiva, se especifica qué se necesita hacer.
- El **método de la tarea** describe cómo se puede realizar una tarea a través de la descomposición en subfunciones, más un régimen de control sobre la ejecución de las subfunciones. En otras palabras, se detalla cómo alcanzar el objetivo.

```
TASK-KNOWLEDGE <identificador del conocimiento sobre la tarea>
    <Descripción de tareas>
    <Descripción de métodos de tareas>
END TASK-KNOWLEDGE <identificador del conocimiento sobre la tarea>
```

Figura 19.13: Estructura del conocimiento sobre tareas en CML.

Para no tener que “reinventar la rueda”, los modelos de conocimiento, en los que se incluyen las tareas, pueden reutilizarse en un nuevo proyecto. Así, las **plantillas de tareas** forman un componente común de combinaciones reutilizables de elementos del modelo, en las que se especifican las inferencias y el conocimiento de la tarea para resolver un problema determinado. Estas plantillas pueden reutilizarse directamente en los SBCs especificando los roles correspondientes en el conocimiento sobre inferencias, o bien hacer alguna modificación para adaptarlas al problema concreto del nuevo sistema.

Existen dos grupos de tareas que se pueden realizar en el proceso de razonamiento intensivo: las tareas de análisis y las tareas de síntesis, y se diferencian por la relación con el sistema sobre el que operan. Las **tareas de análisis** tienen como entrada algunos datos sobre el sistema y producen alguna caracterización del sistema como salida. En las **tareas de síntesis**, el sistema no existe aún, ya que el propósito es construir una descripción del mismo a partir de ciertos requerimientos que debe satisfacer. Se pueden examinar en detalle las plantillas de tareas que propone CK en [Schreiber y otros, 2000] y [Alonso y otros, 2004].

- **Conocimiento sobre Tareas: La tarea**

La tarea define una función de razonamiento complejo, en la que el nivel superior se corresponde con la tarea identificada en la tarea del modelo. La principal diferencia con las funciones tradicionales de IS es que los datos manipulados en la tarea se describen con la finalidad de que sean independientes del dominio y reutilizables.

En la especificación de la tarea (TASK) en CML, en el apartado GOAL se da una descripción textual informal de las metas que se intentan conseguir. En los roles INPUT y OUTPUT no se incluyen roles estáticos ya que éstos están ligados a las inferencias, y no se indican las proyecciones entre los roles y los términos específicos del dominio. La proyección es indirecta: los roles de las tareas se enlazan a los roles de inferencia a través de las estructuras de control. Cada rol de inferencia tiene una proyección asociada a las construcciones del dominio. En

el ejemplo de la Figura 19.14, se especifica una tarea de valoración que consiste en asociar una categoría de decisión a un caso concreto. Aplicado al ejemplo de la central nuclear, la tarea en el sistema va a catalogar un estado (por ejemplo como normal, grave, muy grave, crítico) según los datos de entrada y con unos criterios establecidos. Se puede observar cómo, excepto por los comentarios, no se hace referencia al conocimiento del dominio en los roles.

```

TASK Valoración_de_gravedad;
GOAL: "Valorar el estado del sistema de refrigeración de la central nuclear y catalogarlo según su gravedad"
ROLES:
INPUT:
    Caso: "Valores actuales de los sensores del sistema"
    Criterios_de_valoración: "Criterios para ponderar la gravedad de los valores de los sensores"
OUTPUT:
    Decisión_catalogación: "Catalogación del caso de entrada respecto a la gravedad para el sistema"
END TASK Valoración_de_gravedad;

```

Figura 19.14: Ejemplo de una especificación de tarea en CML.

- **Conocimiento sobre Tareas: El método de la tarea**

En el método de la tarea se identifica la estrategia de razonamiento empleada para resolver un problema. Indica cómo se descompone la tarea en subtareas integradas en una estructura de control que funcionalmente recoge la forma de resolver la tarea. Como se muestra en el ejemplo de la Figura 19.15, en la estructura de control pueden aparecer funciones de programación (como estructuras de bucle, condicionales, etc.), usar roles intermedios para almacenamiento de resultados parciales y llamadas a subtareas, inferencias y funciones de transferencia para, en conjunto, especificar el proceso de resolución de la tarea.

19.5.2.2 El Modelo de comunicación

En el Modelo de Comunicación se describen de forma detallada los procesos de transferencia de información y conocimiento entre el SBC y otros agentes externos, pudiendo ser estos humanos, máquinas, sistemas de información u otros SBCs. Teniendo en perspectiva que las tareas realizadas por los agentes generan productos (elementos de información), en este modelo se especificarán las formas de transmisión tras la consecución de estos productos. El Modelo de Comunicación se compone de tres elementos que se describen a continuación: el plan de comunicaciones, las transacciones y la especificación del intercambio de información.

El Plan de Comunicaciones. En el plan de comunicaciones se detalla el diálogo a alto nivel de dos agentes para llevar una tarea conjunta. La información que se transmite entre los dos agentes estará en parte descrita en los Modelos de Tareas, Agentes y Conocimiento especificadas en las etapas anteriores. Las actividades que deben realizarse para construir este modelo son:

```
TASK-METHOD Valoración_de_gravedad_usando_criterios;
REALIZES: Valoración_de_gravedad;
DECOMPOSITION:
    INFERENCES: Abstraer, Especificar, Seleccionar, Evaluar;
    TRANSFER-FUNCTIONS: ;
ROLES:
    INTERMEDIATE:
        Característica_de_caso: "Uno de los valores de los sensores del caso actual";
        Criterio: "Uno de los criterios especificados en los criterios de entrada";
        Valor: "Valoración parcial del estado actual del sistema";
        Resultados: "Acumulador de los resultados en el proceso de valoración";

    CONTROL-STRUCTURE:
        WHILE NEW-SOLUTION Abstraer( Caso-> Característica_de_caso ) DO
            Caso:= Característica_de_caso UNION Caso;
        END WHILE

        Especificar( Caso -> Criterios_de_valoración );

        REPEAT
            Seleccionar( Criterios_de_valoración -> Criterio);
            Evaluar( Caso + Criterio -> Valor );
            Resultados := Valor UNION Resultados;
        UNTIL HAS-SOLUTION Equiparar( Resultados -> Decisión_catalogación);

END TASK-METHOD Valoración_de_gravedad_usando_criterios;
```

Figura 19.15: Ejemplo de una especificación de un método de tarea en CML.

- Para cada uno de los agentes que intervienen en el sistema, se deben recopilar los documentos de trabajo y funciones de transferencia donde se intercambia información con otro agente.
- A partir de la información anterior, identificar las transacciones entre agentes que se realizan y dar un nombre a cada una de ellas.
- Construir, como se muestra en el ejemplo de la Figura 19.16, el diagrama de diálogo que representa todas las transacciones de información entre dos agentes y las tareas involucradas en el intercambio de información.
- Añadir al diagrama de diálogo un control sobre las transacciones, representándolo mediante un diagrama de transición de estados usando UML o mediante un pseudocódigo propio de CK. Para este último caso, existen un conjunto de primitivas que se muestran en la Tabla 19.13.

Las Transacciones. Una vez identificado el plan de comunicaciones donde se recoge el flujo de información, las listas de transacción y el control de las transacciones, en este apartado debe detallarse cada una de las transacciones que se producen entre dos

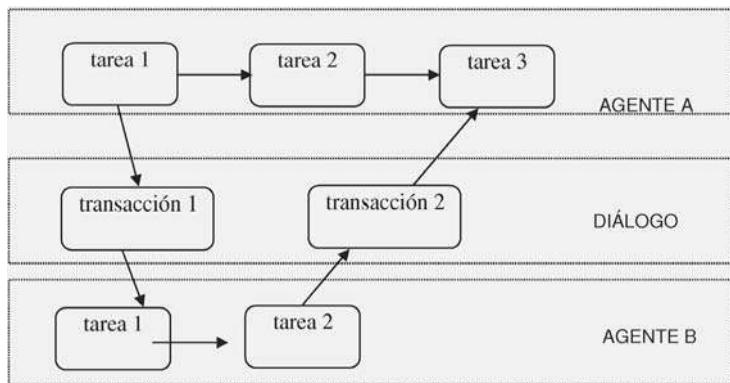


Figura 19.16: Ejemplo de diagrama de diálogo del plan de comunicaciones en CK.

Instrucción	Argumentos	Descripción
SEND	(transacción o mensaje)	operador de comunicación básico
RECEIVE	(transacción o mensaje)	operador de comunicación básico
CARRY-OUT	(transacción)	combinación SEND/RECEIVE
WAIT-until/while	(condición)	acción de espera para comunicaciones
PROCESS	(tarea)	parte de tareas ajenas al modelo de comunicación
;	(transacción1; transacción2)	operador de secuencia
REPEAT until/while	(condición)	instrucción de control básica
IF THEN ELSE	(condición transacción1 transacción2)	instrucción de control básica
&	(transacción1 & transacción2)	operador AND
-	(transacción1 - transacción2)	operador de selección (OR exclusivo)
∨	(transacción1 ∨ transacción2)	operador OR
..		separador para la especificación de control

Tabla 19.13: Primitivas del Modelo de Comunicación en CK.

agentes y dos tareas. Para ello, CK propone el formulario CM-1, que se muestra en la Tabla 19.14 con un ejemplo, que contiene los siguientes campos:

- **Nombre de la transacción:** se asigna un nombre y se comenta brevemente su finalidad.
- **Objetos de información:** identificación del objeto central del intercambio de información e identificación de las tareas entre las que se transmite.
- **Agentes involucrados:** identificación del agente emisor y receptor.
- **Plan de comunicaciones:** se indica de qué plan de comunicaciones forma parte.
- **Restricciones:** precondiciones para que se pueda efectuar la transacción y post-condiciones que pasarán a ser válidas tras realizarse la comunicación.
- **Especificación del intercambio de información:** descripción del tipo de mensajes que componen la transacción. Se puede detallar en los documentos del tipo CM-2.

Modelo de Comunicación	Documento sobre el Modelo de Comunicación (CM-1)
NOMBRE DE TRANSACCIÓN	Aviso de elemento averiado: el SBC ha detectado un componente que no funciona por medio de los sensores. Será el ingeniero que monitoriza el sistema el que deba dar parte al dpto. de mantenimiento y reparaciones.
OBJETOS DE INFORMACION	Código de referencia del componente averiado. Se detecta en el proceso de monitorización del sistema.
AGENTES INVOLUCRADOS	El ingeniero que monitoriza el SBC recoge el aviso del SBC. El ingeniero deberá comunicar la avería al dpto. de mantenimiento.
PLAN DE COMUNICACIONES	Referencia al plan de comunicaciones <i>P_CN2</i> .
RESTRICCIONES	El ingeniero que monitoriza el sistema debe validar el aviso indicando en el ordenador que lo ha recibido.
ESPECIFICACIÓN DEL INTERCAMBIO DE INFORMACIÓN	El tipo de mensaje es: <i>aviso de avería; confirmación de recibido</i> .

Tabla 19.14: Ejemplo de un formulario CM-1 sobre transacciones del Modelo de Comunicación.

La Especificación del Intercambio de Información. Usualmente se suele ampliar el contenido de una transacción entre agentes, más aún cuando ésta puede estar formada por más de un mensaje. Para ello, se amplía en contenido del formulario CM-1 con el formulario CM-2, que corresponde al nivel más detallado del Modelo de Comunicación. En él se especifica la estructura, forma, contenido y medio de transmisión de los mensajes que forman la transacción. En la Tabla 19.15 se muestra un ejemplo de un documento CM-2. Los apartados de este formulario son los siguientes:

- **Transacción:** identificador y nombre de la transacción de la que forma parte, que debe coincidir con la indicada en el formulario CM-1.

- **Agentes involucrados:** identificación del agente emisor y receptor.
- **Items de información:** formado por una lista con todos los elementos que se van a transmitir en la información. Se debe indicar el *rol*: esto es si es el objeto central de información o si es información de soporte; la *forma*: la estructura sintáctica de transmisión; y el *medio*: el canal por el que se realiza la comunicación.
- **Especificación de los mensajes:** se debe describir, para cada mensaje que forma la transacción, el *tipo de comunicación*: si es del tipo pregunta/respuesta, informe, oferta/propuesta, etc. (existen tipos definidos en [Schreiber y otros, 2000]); el *contenido*: la proposición o declaración que contiene; y la *referencia*: si es necesario, indicar el modelo del dominio al que pertenece.
- **Control sobre mensajes:** se indican las operaciones de control sobre el flujo de mensajes. Puede estar formado por las instrucciones usadas en el plan de comunicaciones indicadas en la Tabla 19.13.

Modelo de Comunicación	Documento sobre el Modelo de Comunicación (CM-2)
TRANSACCIÓN	Aviso de elemento averiado
AGENTES INVOLUCRADOS	<i>Emisor</i> : SBC. Aviso por pantalla de que posiblemente un elemento del circuito no funciona. <i>Receptor</i> : ingeniero del dpto. de refrigeración. Debe confirmar la recepción del mensaje. Si lo catalogara como avería, debe informar al dpto. de mantenimiento. En caso contrario, debe elaborar un informe del falso aviso.
ITEMS DE INFORMACION	<i>Rol</i> : el aviso del SBC es un objeto central de información. <i>Forma</i> : código de referencia del componente, descripción de los atributos y causas posibles de la avería. <i>Medio</i> : mensaje visible en la pantalla de monitorización del sistema.
ESPECIFICACIÓN DE LOS MENSAJES • comunicación de avería • confirmación de recibido	<i>Tipo</i> : informe <i>Contenido</i> : referencia de elemento, descripción de elemento, hora de detección, lista de causas posibles de avería. <i>Tipo</i> : respuesta <i>Contenido</i> : el ingeniero debe dar su código en el sistema para identificarlo.
CONTROL DE LOS MENSAJES	<i>Envío de comunicación de avería; recepción de confirmación de recibido</i>

Tabla 19.15: Ejemplo de un formulario CM-2 sobre una especificación del intercambio de información del Modelo de Comunicación.

19.5.3 Modelado artefactual en CommonKADS

Una vez especificado el Modelo Conceptual, CK propone una serie de pasos y técnicas para abordar el desarrollo del SBC. Este modelado artefactual o modelado a nivel de implementación consiste en la construcción de un único modelo que irá recogiendo las especificaciones del producto software final, denominado *Modelo de Diseño*. El proceso de diseño del sistema debe basarse en las restricciones y requisitos especificados en los análisis de los anteriores Modelos de la Organización, Tareas, Agentes,

Conocimiento y Comunicación, especialmente en estos dos últimos. La metodología CK recomienda seguir el principio de conservación de la estructura, que establece que los contenidos existentes en los modelos de análisis se deben mantener durante el diseño y la implementación. Cualquier cambio que se decida en el diseño debe reflejarse explícitamente en los demás modelos del sistema.

El proceso de desarrollo de SBCs persigue las directrices y técnicas usadas en IS, como son la separación del análisis de la implementación, la reutilización o las garantías de calidad. Además, sigue uno de los principios de MDA (*Model-Driven Architecture*)², que considera el modelo de diseño como una especificación independiente de la plataforma de implementación. Así, el Modelo de Diseño no crea como producto de salida un sistema implementado sino que establece las directrices de cómo debe hacerse.

En CK se proponen una serie de pasos consecutivos que deben seguirse para completar el Modelo de Diseño, empezando con la definición de la arquitectura software en función de sus módulos y subsistemas y, basada en ella, la selección del hardware y software que se va a utilizar para la implementación. Con estos condicionantes, a continuación se especifican en detalle los subsistemas y componentes definidos en la arquitectura, completando la descripción de las interfaces y la funcionalidad que deben tener los módulos del sistema. Por último, los elementos de los módulos anteriores de análisis deben proyectarse en la arquitectura de referencia. A continuación se comentarán muy brevemente estos pasos que constituyen el Modelo de Diseño de CK. Se recomienda al lector que, para comprender mejor esta parte del modelado y ver ejemplos del diseño, lea [Schreiber y otros, 2000] y [Alonso y otros, 2004].

19.5.3.1 Diseño de la arquitectura del sistema

La meta de este paso es la identificación de los subsistemas que componen la arquitectura del SBC, y la definición de los elementos de control y comunicación entre ellos. CK recomienda utilizar una arquitectura de referencia que se utilizaría para definir la arquitectura global del sistema y la arquitectura en el modelo de la aplicación.

Arquitectura global del sistema. Aunque pueden elegirse otros paradigmas, CK recomienda utilizar la arquitectura Modelo-Vista-Controlador (MVC) [Goldberg, 1990] que consiste en especificar, en primer lugar el *modelo de la aplicación* que contendrá las funciones de razonamiento, los datos de la base de conocimiento y los roles dinámicos del sistema. A continuación, se definirían las *vistas* externas de las funciones y los datos incluidos en el modelo de la aplicación. Éstas representan las visualizaciones para las interfaces de usuario, pero también los datos que pueden ser consultados en cada caso contra una base de conocimiento. Por último se diseñaría el *controlador* del sistema formado usualmente por un conjunto de gestores de eventos que los procesarían según las directrices especificadas. Aquí es donde se implementaría las transacciones que se indican en el Modelo de Comunicación.

²<http://www.omg.org/mda/papers.htm>

Arquitectura del modelo de la aplicación. En el modelo de la aplicación se definen las funciones de razonamiento (tareas e inferencias) y las estructuras de información y conocimiento (el conocimiento del dominio). En la arquitectura que propone CK, se especifican las relaciones de flujos de acceso de estos elementos y la invocación de los métodos. Se incorporan, además, especificaciones de datos o funciones asociadas a estos elementos. Por ejemplo, en una inferencia se deberá especificar en el diseño, los algoritmos que puede utilizar en su invocación. En la Figura 19.17 se muestra un esquema de la arquitectura del modelo de aplicación que propone CK, cuyos componentes serán descritos en la sección 19.5.3.3. En la figura, las líneas continuas muestran los posibles flujos de acceso de información entre los elementos, y las líneas discontinuas indican las posibilidades de invocación de ejecución a funciones y métodos.

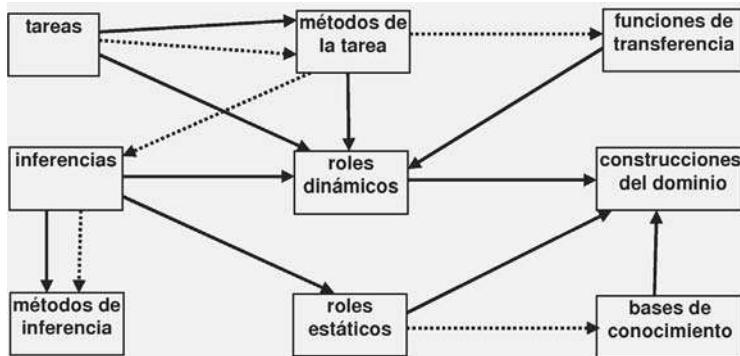


Figura 19.17: Esquema de la arquitectura del modelo de aplicación que propone CK.

Para representar la arquitectura del sistema, CK ofrece un formulario DM-1 que contendrá los siguientes campos:

- **Organización de los subsistemas:** en este apartado se hace referencia al diagrama de subsistemas, usando estructuras estándares como la mencionada MCV, máquinas abstractas, diagramas de pizarras, etc.
- **Modelo de control:** aquí se hace referencia al régimen de control interno de los módulos, como orientado a eventos, centralizado, por petición, etc.
- **Descomposición de los subsistemas:** se referencia a los diagramas en los que se descomponen los subsistemas como el que se muestra en la Figura 19.17, y se debe especificar el paradigma elegido para la descomposición, como orientada a objetos, funcional, declarativo, etc.

19.5.3.2 Selección de la plataforma de implementación

En este apartado, el equipo de desarrollo debe seleccionar la plataforma hardware y software en la que va a construir el SBC. Actualmente, el hardware y el software

forman sólo una pequeña parte de los costes totales de un proyecto, ya que los equipos informáticos actuales de grandes prestaciones son cada vez más asequibles y existen herramientas de software libre de representación del conocimiento como Protege³ y sus *plugins* que abaratan mucho los costes en este apartado. Por este motivo, la selección debe centrarse más en conseguir buenas prestaciones de calidad del sistema y ahorros de tiempos de desarrollo que en otros factores. CK propone la hoja de trabajo DM-2 que recoge características de las plataformas de implementación para poder compararlas y tomar una decisión de selección. Las características que se proponen en DM-2 son las siguientes:

- **Producto software:** identificación del nombre y versión del producto.
- **Hardware potencial:** sistema hardware necesario para instalarse.
- **Librería de visualización:** se debe estudiar si dispone de librerías adecuadas para visualización y si son adecuadas para las necesidades de desarrollo del SBC.
- **Lenguaje de implementación:** se analiza si los lenguajes de la plataforma permiten fácilmente expresar las necesidades software del sistema.
- **Representación del conocimiento:** se analiza si la plataforma está diseñada para representar el conocimiento necesario en el SBC, si permite la representación de conocimiento declarativo, axiomas, reglas, etc.
- **Protocolos de interacción:** en el caso de que el SBC necesite comunicarse con otros sistemas, debe estudiarse las facilidades en el uso de los protocolos de procesamiento distribuido.
- **Control de flujo:** se debe estudiar si la plataforma permite utilizar un mecanismo basado en el paso de mensajes y si es posible crear varios hilos.
- **Soporte para CommonKADS:** se ganaría en tiempo de desarrollo si la plataforma dispusiera de soporte para el modelado de los modelos de análisis de CK y, por ejemplo, integrara las arquitecturas de referencia que propone CK.

19.5.3.3 Especificación de los componentes de la arquitectura

En el tercer paso del Modelo de Diseño se establecerá la arquitectura del sistema de forma más detallada. Aquí se deberán definir de forma concreta las interfaces entre los subsistemas y módulos, y las funcionalidades de estos elementos. Se recogerá en el documentos de trabajo DM-3, que contienen los elementos indicados en la arquitectura global del sistema y los que aparecen en el modelo de la aplicación que aparecen en la Figura 19.17:

- **El controlador:** se establecerán los mecanismos para la gestión de eventos internos y externos. En este componente se va a implementar las especificaciones del Modelo de Comunicación. Se deberán indicar la forma de las interfaces de

³<http://protege.stanford.edu>

comunicación de eventos, el diseño del reloj interno para tratar interrupciones, si se necesita ejecución concurrente de procesos, si hay comunicación con otros sistemas, etc. El diseño del controlador debe hacerse de forma que contemple todas las eventualidades que pueden surgir en la ejecución del sistema, como el control de los tiempos de los eventos, de los procesos de razonamiento, o las esperas de señales de componentes hardware.

- **Las tareas:** se debe indicar cómo se activa y qué parámetros de entrada necesita, y cómo la tarea invoca al método de ejecución.
- **Los métodos de las tareas:** para describir el método de la tarea, se debe decidir el lenguaje de implementación que se va a utilizar para representarlo y dónde se va realizar esta implementación que dependerá del paradigma del lenguaje de programación que se haya seleccionado.
- **Las inferencias:** se debe indicar cuál es el diseño interno de la inferencia con respecto a los métodos de inicialización, así como la gestión de la memoria interna de trabajo usada en el proceso de razonamiento.
- **Los métodos de las inferencias:** se diseñan los métodos de inferencia bajo el punto de vista de la implementación, como decidir el tipo de encadenamiento, métodos de equiparación, y demás algoritmos de razonamiento. El diseño debe hacerse con el propósito de reutilización, ya que un método, situado en una librería de métodos, debería poder invocarse desde distintas inferencias. De esta forma, aquí deben especificarse los parámetros de entrada de forma que en el método se asocien a los correspondientes roles dinámicos y estáticos del sistema.
- **Los roles dinámicos:** deben declararse para cada rol los tipos de datos, o conceptos de la jerarquía de conceptos, que puede tomar el rol. También se decide aquí los métodos de acceso y modificación de estos tipos de datos o conceptos.
- **Los roles estáticos:** se debe indicar el diseño de las funciones de acceso a los elementos del sistema. Pueden diseñarse para que devuelva una sola instancia del elemento, varias o todas, o que indique si existe o no alguna instancia.
- **Las bases de conocimiento:** se debe establecer la forma de representar las instancias de las reglas, dependiendo de las necesidades del sistema. Se define cómo se actualiza el contenido de las bases de conocimiento, por ejemplo, para mantener la consistencia. Debe determinarse también cómo se organizan los ficheros que contendrán las bases de conocimiento del sistema.
- **Las construcciones del dominio:** aquí se hace referencia a otros elementos del sistema que no aparecen en los apartados anteriores como las definiciones de conceptos, relaciones y tipos de reglas. No necesitan funciones adicionales por lo que sólo es necesario indicar el lenguaje y formato de representación de estos.

- **Las vistas:** en este apartado debe determinarse, por un lado, qué información va a estar accesible desde cada uno de los elementos que componen el sistema y, por otro, deben diseñarse las interfaces de los módulos tanto para la comunicación entre ellos, entre otros sistemas, y la apariencia y funcionalidades de las interfaces de comunicación con los usuarios finales.

19.5.3.4 Especificación de la aplicación sobre la arquitectura

El último paso consiste en proyectar los elementos específicos de la aplicación en la arquitectura seleccionada. Estos se realizan en las dos fases siguientes:

1. Proyectar la Información de los Modelos de Análisis:

La información obtenida en el análisis, se relaciona en la arquitectura, asociando cómo estarán vinculados cada uno de los elementos con los componentes de la aplicación software. Si partimos de la arquitectura de referencia de la sección anterior, esta fase consiste en crear instancias de los elementos correspondientes (tareas, métodos de tareas, etc.). Se debe tener presente que, como recomienda CK, se debe seguir el principio de conservación de la estructura, de modo que cualquier cambio significativo que se decida en el diseño debe quedar reflejado en los modelos de análisis.

2. Añadir Detalles Específicos de Diseño:

Para cada uno de los componentes del modelo de la aplicación, exceptuando aquellos que estaban perfectamente definidos en el paso anterior (tareas, roles estáticos y construcciones del dominio), deben indicarse las decisiones del diseño que van a incorporarse a la aplicación implementada. CK propone utilizar el documento de trabajo DM-4 para recoger estas especificaciones, que deberá incluir información sobre:

- **El controlador:** debe indicarse con detalle cómo se implementa el control del plan de comunicaciones y las transacciones en gestores de eventos.
- **Los métodos de las tareas:** se debe formalizar la estructura de control en el lenguaje proporcionado en la plataforma de implementación seleccionada.
- **Las inferencias:** se debe especificar cómo se van a invocar los métodos de inferencias, en concreto cómo se van a asociar los correspondientes roles a los argumentos del método.
- **Los métodos de las inferencias:** se deben construir o seleccionar desde un repositorio, los métodos de inferencia que se asocian a las inferencias.
- **Los roles dinámicos:** los tipos de datos que se habían especificado para cada rol, deben asociarse con los tipos de datos existentes en la plataforma y lenguaje seleccionado. Es necesario que el lenguaje tenga la suficiente capacidad expresiva como para recoger las necesidades indicadas en la especificación de los componentes. Si no, habría que volver a estudiar las decisiones que se tomaron en el paso segundo del Modelo del Diseño.

- **Las bases de conocimiento:** se debe indicar cómo representar el formato de instancias establecido en las bases de conocimiento en el lenguaje proporcionado por la plataforma de desarrollo.
- **Las vistas:** siguiendo los principios de la IS para el diseño de interfaces, deben establecerse las vistas para cada uno de los componentes del sistema, intentando utilizar las proporcionadas por la plataforma de desarrollo seleccionada.

Los métodos y técnicas vistos en este capítulo sirven para abordar todas las fases de construcción de un SBC, desde la identificación del problema hasta el diseño de la implementación. En cada fase se ha aconsejado que, siempre que sea posible, el InCo debe analizar los componentes ya existentes para intentar incorporarlos a su sistema y, si esto no es posible y debe desarrollarlos, su diseño debe efectuarse de forma que estos nuevos elementos sean fácilmente reutilizables en otros sistemas.

Actualmente, con la filosofía de las arquitecturas orientadas a servicios (SOA *Service-Oriented Architecture*) [Weerawarana y otros, 2005], estos procesos de reutilización se realizan de forma más habitual, incluyendo los casos de desarrollo de SBCs [Bussler y otros, 2002]. La idea se basa en la interconexión de componentes, que realizan tareas parciales de representación de conocimiento y resolución de problemas, comunicados entre sí para resolver un problema. De esta forma, los nuevos SBCs están formados por componentes interrelacionados que siguen estándares de datos y protocolos de comunicación, con mayor o menor capacidad de razonamiento, y que cooperan entre sí para obtener una solución. Con ello, se consigue reducir los costes y tiempos de desarrollo de los SBCs e incrementar sus capacidades.

19.6 Lecturas recomendadas

La bibliografía que trata la Ingeniería del Conocimiento y los Sistemas Expertos es muy amplia. A menudo, el contenido de esta disciplina se incluye en temas más o menos extensos en libros de IA de propósito general como puede ser el muy conocido de [Russell y Norvig, 2004]. Si el lector quiere ampliar conocimientos específicos de Ingeniería del Conocimiento, y estudiar técnicas de representación del conocimiento y de adquisición del conocimiento, se recomienda la lectura de [Gómez y otros, 1997] y [Alonso y otros, 2004]. Si se quieren ampliar los procesos, técnicas y ejemplos completos de la metodología CommonKADS, se recomienda la lectura de [Schreiber y otros, 2000] y [Alonso y otros, 2004].

19.7 Resumen

En este capítulo se ha realizado una visión general de la Ingeniería del Conocimiento como rama de la IA para el desarrollo de Sistemas Basados en Conocimiento. Se mencionan algunas técnicas de adquisición del conocimiento y se enumeran distintos métodos para el desarrollo de SBC, pero la mayor parte del capítulo se ha querido

dedicar a la descripción de los procesos de la metodología CommonKADS para ofrecer al lector una idea en conjunto de las fases que conlleva el desarrollo de un SBC.

19.8 Ejercicio propuesto

19.1. Realizar la documentación para el desarrollo de un SBC con tema libre siguiendo la metodología CommonKADS completando todas sus hojas de trabajo.

Referencias

- ALBERICO, R. y MICCO, M.: *Expert Systems for Reference and Information Retrieval*. Miocler Corporation, 1990.
- ALONSO, A.; GUIJARRO, B.; LOZANO, A.; J., PALMA y TABOADA, M.J.: *Ingeniería del Conocimiento. Aspectos metodológicos*. Pearson Prentice Hall, 2004.
- ANGELE, J.; FENSEL, D. y STUDER, R.: «Domain and Task Modelling in MIKE». En: A. Sutcliffe et al., eds., *Domain Knowledge for Interactive System Design*, Chapman and Hall, 1996.
- ANJEWIERDEN, A.: «CML2». *Informe técnico 11*, University of Amsterdam, 1997. Disponible en: [<http://www.swi.psy.uva.nl/usr/anjo/cml2>].
- BARBERO, M.I.: *Psicometría II: Métodos de Elaboración de Escalas*. UNED, 1993.
- BARKER, V. y O'CONNER, D.: «Expert Systems for Configuration at Digital: XCON and Beyond». *Communications of the ACM*, 1989, **32(3)**, pp. 298–317.
- BOEHM, W.: «A Spiral Model of Software Development and Enhancement». *IEEE Computer*, 1988, **21(5)**, pp. 61–72.
- BOOCHE, G.; JACOBSON, I. y RUMBAUGH, J.: «The UML Specification Documents». *Informe técnico*, Rational Software Corp., Santa Clara. CA, 1997.
- BUCHANAN, B.; BARSTOW, D.; BECHTEL, R.; BENNETT, J.; CLANCEY, W.; KULIKOWSKI, C.; MITCHELL, T. y D.A., WATERMAN: «Constructing an expert system». En: F. Hayes-Roth; D.A. Waterman y D.B. Lenat (Eds.), *Building Expert Systems*, pp. 127–167. Addison-Wesley, Reading, Mass, 1983.
- BUCHANAN, B.G. y SHOTLIFFE, E.H.: *Rule-Based Expert Systems: The MYCIN Experiment of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1984.
- BUCHANAN, B.G.; SUTHERLAND, G.L y E.A., FEIGENBAUM: «Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry». En: B. Meltzer y D. Michie (Eds.), *Machine Intelligence*, volumen 4, pp. 209–254. Edinburgh University Press, 1969.
- BUSSLER, C.; MAEDCHE, A. y FENSEL, D.: «A Conceptual Architecture for Semantic Web Enabled Web Services». *ACM Special Interest Group on Management of Data*, 2002, **31(4)**, pp. 24–29.
- CARRIZO, M.A.; GIRAD, J.E. y JONES, J.P.: *Building Knowledge Systems: Developing and Managing Rule-Based Applications*. McGraw-Hill, 1989.
- DAVENPORT, T. y PRUSAK, L.: *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press. New York, 1999.

- DE HOOG, R.; MARTILO, R.; WIELINGA, B.; TAYLOR, R.; BRIGHT, C. y VAN DE VELDE, W.: «The CommonKADS Model Set». *Informe técnico ESPRIT Project P5248*, University of Amsterdam, 1994.
- DUDA, R; HART, P.; NILSSON, N.J. y REBOH, R.: *A Computer Based Consultant for Mineral Exploration*. SRI International, 1979.
- EDWARDS, J.S.: *Building Knowledge-Based Systems. Towards a Methodology*. Pitman Publishing, London. UK, 1991.
- FIKES, R.E. y NILSSON, N.J.: «STRIPS. A New Approach to the Application of Theorem Proving to Problem Solving». *Artificial Intelligence*, 1971, **2(34)**, pp. 189–208.
- GOLDBERG, A.: «Information Models, Views and Controllers». *Dr Dobbs Journals*, 1990, **July 1990**, pp. 54–61.
- GUIDA, G. y TASSO, C.: *Design and Development of Knowledge-Based Systems: from Life Cycle to Development Methodology*. John Wiley and Sons, Chichester, UK, 1994.
- GÓMEZ, A. y BENJAMINS, R.: «Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods». En: *Proceedings of the IJCAI'99. Workshop on Ontologies and PSMs. Stockholm. Sweden*, , 1999.
- GÓMEZ, A.; JURISTO, N.; MONTES, C. y PAZOS, J.: *Ingeniería del Conocimiento*. Editorial Centro de Estudios Ramón Areces, 1997.
- HARMON, P. y KING, D.: *Expert Systems*. John Wiley & sons, Inc. EEUU, 1985.
- LINSTER, M. y MUSEN, M.A.: «Use of KADS to Create a Conceptual Model of the ONCOCIN Task». *Knowledge Acquisition*, 1992, **4(1)**, pp. 55–85.
- LINSTONE, H.A. y TUROFF, M.: *The Delphi Method*. Addison-Wesley, 1987.
- MITCHELL, T.M.: «Generalization as Search». *Artificial Intelligence*, 1982, **18(2)**, pp. 203–226.
- NECHES, R.; FIKES, R.; FININ, T.; GRUBER, T.; PATIL, R.; SENATOR, T. y SWARTOUT, W.: «Enabling Technology for Knowledge Sharing». *AI Magazine*, 1991, **12(3)**, pp. 36–56.
- NEWELL, A.: «The Knowledge Level». *Artificial Intelligence*, 1982, **18**, pp. 87–127.
- NEWELL, A. y SIMON, H.: *GPS, A Program that Simulates Human Thought*. McGraw-Hill, NY, 1963.
- OSBORN, A.F.: *Applied Imagination*. Charles Scribner's Sons, 1953.
- PAJARES, G. y SANTOS, M.: *Inteligencia Artificial e Ingeniería del Conocimiento*. RA-MA, 2005.

- PARSAYE, K. y CHIGNELL, M.: «Expert Systems for Experts». En: *Measuring Expert Systems Performance*, pp. 365–374. John Wiley & sons, 1988.
- QUINLAN, J.R.: «Induction of decision trees». *Machine Learning*, 1986, **1**(1), pp. 81–106.
- ROYCE, W.W.: «Managing the Development of Large Software Systems». En: *Proc. of the IEEE westcon. CA*, IEEE Press, 1970.
- RUSSELL, S. y NORVIG, P.: *Inteligencia Artificial. Un Enfoque Moderno. Segunda edición*. Pearson Educación, 2004.
- SACILE, R.: «Using CommonKADS to Build an Expertise Model for Breast Cancer Prognosis and Therapy». *Informe técnico RR-2737*, INRIA-Sophia Antipolis, 1995.
- SCHREIBER, G.; AKKERMANS, H.; ANJEWIERDEN, A.; DE HOOG, R.; SHADBOLT, N.R.; VAN DE VELDE, W. y WIELINGA, B.J.: *Knowledge Engineering and Management. The CommonKADS Methodology*. MIT Press. Massachussets, 2000.
- TABOADA, M.; LAMA, M.; BARRO, S.; MARIN, R.; MIRA, J. y PALACIOS, J.: «A Problem-Solving Method for “Unprotocolised” Therapy Administration Task in Medicine». *Artificial Intelligence in Medicine*, 1999, **17**, pp. 157–180.
- WATERMAN, D.A.: *A Guide to Expert Systems*. Addison-Wesley. MA. EEUU, 1986.
- WEERAWARANA, S.; CURBERA, F.; LEYMANN, F.; STOREY, T. y FERGUSON, D.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more*. Prentice-Hall, 2005.
- WIELINGA, B.J.; SCHREIBER, A.T. y BREUKER, J.A.: «KADS: A Modelling Approach to Knowledge Engineering». *Knowledge Acquisition*, 1992, **4**, pp. 5–53.
- WINSTON, P.H.: «Learning structural descriptions from examples». *Informe técnico AI-TR-231*, MIT. Cambridge, Mass, 1970.

Capítulo 20

Sistemas multiagentes

Juan Pavón Mestras
Universidad Complutense de Madrid

20.1 Introducción

A lo largo del libro se han ido presentando diversos métodos y técnicas de IA. La cuestión que se plantea en este capítulo es cómo integrarlos en la construcción de sistemas reales. Hasta principios de los años noventa, los sistemas *inteligentes* solían ser programas aislados, orientados a resolver problemas muy concretos. La complejidad residía principalmente en el algoritmo y no tanto en la arquitectura del sistema. Pero en la actualidad la complejidad está en la diversidad de componentes que forman los sistemas. Estos componentes proporcionan servicios, que se pueden combinar de múltiples maneras para crear nuevos servicios, más versátiles, que muchas veces se pueden personalizar para cada usuario. Además, los componentes están distribuidos en varias máquinas, dispersas geográficamente a través de redes fijas o inalámbricas. Ocurre también que los dispositivos sobre los que se ejecutan son muy heterogéneos, con distintas capacidades, desde teléfonos móviles o PDAs hasta potentes servidores. Todo ello hace que la complejidad de la arquitectura de estos sistemas sea mayor, al tener que abordar cuestiones como la integración de los distintos componentes, cómo cooperan, cómo se instalan, cómo se configuran, cómo entran o salen del sistema, etc. Para tratar estos problemas ha tenido que evolucionar la ingeniería del software. Concretamente, aplicada al campo de la IA, se ha definido el concepto de *agente* para referirse a los componentes distribuidos que forman estos sistemas. La visión de un sistema como un conjunto de agentes que colaboran entre sí para lograr unos objetivos comunes facilita la estructuración y gestión de la complejidad y la heterogeneidad.

Generalmente se pueden ver los agentes como una extensión del paradigma de objetos distribuidos, donde las dos características distintivas fundamentales son *autonomía* y *sociabilidad*.

La autonomía viene determinada por la naturaleza intencional de los agentes: los agentes son entidades que persiguen objetivos. Mientras que un objeto es una entidad pasiva que ejecuta los métodos que le solicite un cliente, un agente tiene la capacidad de decidir por sí mismo qué tareas ejecutar, en función de los objetivos que

quiera lograr. Hasta tal punto esto es así que un agente puede incluso rechazar una solicitud si considera que le aleja del cumplimiento de sus objetivos. Esto se refleja en una frase ya clásica de Wooldridge [Wooldridge y Jennings, 1995]: *Los objetos lo hacen gratis, los agentes porque quieren*. Un ejemplo que ilustra este aspecto es un buscador de información en la web. Lo habitual es que el usuario consulte un motor de búsqueda con varios términos y luego vaya examinando con su navegador una por una las páginas que se le han ofrecido para ver cuáles son más idóneas para sus intereses. Es el usuario quien va guiando todos los pasos que se van realizando. Si utilizara un agente inteligente de búsqueda, podría plantearle los términos y sería el agente quien invocaría al motor de búsqueda y exploraría las páginas para ver cuáles se adecúan más al perfil del usuario. Al cabo de un tiempo devolvería al usuario resultados de más calidad: en vez de varios miles de referencias, sólo una docena de páginas, por ejemplo. Esto requiere que el agente vaya aprendiendo cuáles son las preferencias y las necesidades del usuario para afinar la búsqueda y selección de páginas. Llevado a un extremo, el agente podría anticiparse al usuario y buscar los documentos relacionados con un documento que el usuario está escribiendo o leyendo en un momento determinado, y sugerírselos como información complementaria. Para gestionar la autonomía, los agentes utilizan un conjunto de técnicas de IA como las que se describen en este libro para representar y gestionar el conocimiento (véase los capítulos de la parte II), diagnosticar, razonar, planificar y aprender (véase los capítulos de las partes III y IV). Precisamente, uno de los problemas que se abordan en este tema es cómo integrar estas técnicas para la construcción de los agentes, en la sección 20.2.

Por otra parte, los agentes son entidades sociales. Habitualmente los agentes requieren la colaboración de otros agentes para llevar a cabo sus planes. Por ello, se habla de *sistemas multiagentes*, o SMA para abbreviar. En el ejemplo anterior, nuestro agente de búsqueda avanzado podría recibir ayuda de otros agentes, por ejemplo para interpretar páginas web con formatos particulares, o para consultar si otros agentes de usuarios con perfiles similares han encontrado información relevante de un tema anteriormente y así ahorrarse trabajo. La colaboración de los agentes en un SMA requiere una organización y mecanismos de interacción. La organización determina la arquitectura global del sistema. Define estructuras organizativas, que determinan cómo se agrupan los agentes, utilizando distintos criterios: funcional, geográfico, etc. La organización también se puede ver desde una perspectiva más dinámica como un conjunto de flujos de trabajo que determinan cómo intervienen los distintos agentes para implementar los servicios del SMA. Además, la organización establece un conjunto de normas que los agentes deben cumplir para el desarrollo eficaz de su colaboración. Todos estos aspectos de organización de un SMA se desarrollan en la sección 20.3.

También la interacción entre los agentes extiende el modelo de objetos, y va más allá del simple paso de mensajes. La interacción entre agentes se define en el nivel de conocimiento, añadiendo una componente intencional. Esta comunicación requiere el uso de ontologías (véase el capítulo 5) y una teoría de actos del habla, que se describe en la sección 20.4.

Con todos estos elementos se puede construir un SMA. Pero esto no es en principio una tarea fácil porque los sistemas donde se aplican son complejos y el conjunto de

técnicas que hay que integrar es variado. Las aportaciones al campo de los SMA son multidisciplinares. Así veremos cómo la Sociología, la Psicología o la Biología han contribuido en teorías de organización de SMA o en el diseño del comportamiento de los agentes. Para sistematizar la ingeniería de este tipo de sistemas se están investigando métodos y herramientas de desarrollo de SMA, especialmente para análisis y diseño. No se trata de definir nuevos lenguajes de programación para agentes (aunque algunas propuestas hay) ya que casi todo el software se programa con lenguajes orientados a objetos, especialmente Java con extensiones para definir reglas, razonamiento lógico y planificación. Por tanto, el concepto de agente se trabaja más a nivel de modelo y luego se implementa utilizando técnicas más convencionales. Aunque hay muchas propuestas de lenguajes de modelado de agentes, casi una por metodología, actualmente empieza a haber cierta tendencia unificadora, de la misma manera que UML lo supuso para el modelado con objetos, lo cual permitirá definir estándares y facilitar la difusión y aceptación de la tecnología. En la sección 20.5 se presentan métodos y herramientas actualmente disponibles para construir SMA.

Teniendo en cuenta lo anterior y viendo el tipo de aplicaciones que se desarrollan utilizando agentes, descritas en la sección 20.6, se puede concluir que el principal interés del paradigma de agente para desarrollo de software está en su capacidad para modelar sistemas con conceptos más cercanos a los que suelen utilizarse al tratar sistemas sociales y humanos. De hecho, gran parte de las contribuciones para el desarrollo de esta tecnología provienen de campos tan diversos como la Sociología, la Biología, o la Psicología, aparte de las prácticas de la Ingeniería del Software, la Programación Distribuida y la IA en general. El salto cualitativo que representa el concepto de agente respecto al de objeto es importante. Aparte de las características mencionadas anteriormente, al tratar con agentes aparecen conceptos como estado mental, creencias, objetivos, normas sociales, actos del habla, ontologías, etc., que proporcionan una mayor riqueza expresiva para el modelado de los sistemas que estarán presentes en nuestro entorno en el futuro próximo.

20.2 Arquitecturas de agentes

Para construir agentes, primero hay que diseñarlos de acuerdo a ciertos principios arquitectónicos. Esto quiere decir que hay que definir una arquitectura, esto es, determinar y estructurar sus componentes y las relaciones entre ellos. No hay una arquitectura única para agentes ya que su tipología es muy diversa. De hecho, no hay una acepción única establecida para el término agente y varía generalmente dependiendo del ámbito de aplicación. En la primera Escuela de Verano Europea de SMA (EASSS'99), organizada por AgentLink, donde se dieron cita varios de los expertos más reconocidos en la materia, hubo ocasión de comprobar la disparidad de criterios. Así se refleja en las actas [Weiss et al., 1999]:

- Michael Wooldridge: un agente es un sistema computacional encapsulado, situado en un entorno y con capacidad de actuar autónomamente en ese entorno con el propósito de lograr satisfacer sus objetivos de diseño.

- Michael Huhns: una entidad computacional activa con: (a) identidad persistente, (b) que puede percibir, razonar acerca de, e iniciar actividades en su entorno, (c) que puede comunicarse (con otros agentes).
- Stephan Covaci: un programa que actúa autónomamente en representación de una persona u organización y que tiene su propio flujo de ejecución.
- Henry Lieberman: (un agente de interfaz es) software que es autónomo, aprende, es sensible al contexto, presenta una apariencia humana, tiene conocimiento especializado... Un agente no tiene por qué satisfacer todo esto...

De estas definiciones, y otras que aparecen en la literatura podemos identificar varias propiedades que caracterizan a los agentes:

- *Autonomía*: Los agentes pueden trabajar sin la intervención directa del usuario y tienen cierto control sobre sus acciones y estado interno.
- *Reactividad*: Los agentes pueden percibir su entorno (que puede ser el mundo físico, un usuario detrás de una interfaz gráfica, aplicaciones en la red, u otros agentes) y responder oportunamente a cambios que se produzcan en el mismo.
- *Iniciativa*: El comportamiento de los agentes está determinado por los objetivos (metas) que persiguen y por tanto pueden producir acciones no sólo como respuesta al entorno.
- *Habilidad social*: Para satisfacer sus objetivos un agente puede requerir la colaboración de otros agentes en quienes delega o con quienes comparte la realización de tareas. Para ello necesita coordinarse, negociar, en definitiva, interactuar.
- *Razonamiento*: Un agente puede decidir qué objetivo perseguir o a qué evento reaccionar, cómo actuar para conseguir un objetivo, o suspender o abandonar un objetivo para dedicarse a otro.
- *Aprendizaje*: El agente puede adaptarse progresivamente a cambios en entornos dinámicos mediante técnicas de aprendizaje.
- *Movilidad*: En determinadas aplicaciones puede ser interesante permitir que los agentes puedan migrar de un nodo a otro en una red preservando su estado en los saltos entre nodos.

Para definir la arquitectura interna de los agentes, hay que tener en cuenta inicialmente los aspectos de autonomía y reactividad que aparecen en todos los modelos de agente. Así, en una primera aproximación, todo agente es una entidad que percibe eventos de su entorno y que puede actuar sobre él. Esto se refleja en la Figura 20.1.

El agente tiene una visión parcial del entorno, limitada por sus sensores, que le permiten obtener información sobre determinados eventos. Si se trata de un agente robótico, los sensores pueden ser cámaras u otros dispositivos para ver obstáculos en su camino, pero a lo mejor no tiene dispositivos auditivos que le permitieran captar

sonidos del entorno. Su capacidad de actuar en el entorno también está limitada, dependiendo del tipo de actuadores. En el caso del agente robótico, dispondrá de motores y ruedas o brazos mecánicos para moverse por el entorno o manipular objetos. Un agente web, sin embargo, tendrá otro tipo de sensores y actuadores, ya que tendrá que acceder a servicios web utilizando determinados protocolos de comunicación.

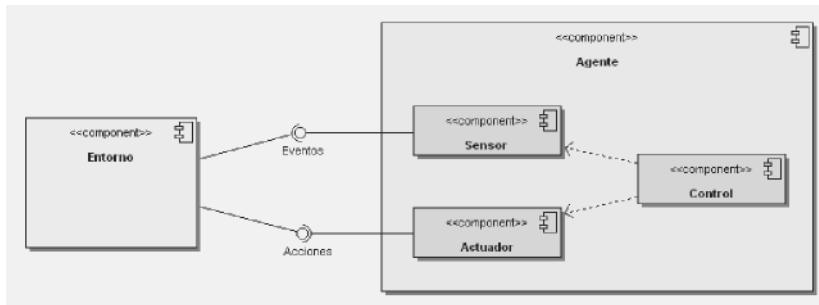


Figura 20.1: Arquitectura general de un agente.

Esta percepción y actuación limitada determina que la representación del mundo que tiene un agente es parcial, lo cual plantea un cierto grado de incertidumbre. El agente tiene que decidir qué acciones tomar teniendo en cuenta el modelo del mundo que va construyéndose a partir de la información que consigue por los sensores. Así, la arquitectura del agente, especialmente el componente de control, es una arquitectura software de un sistema de toma de decisiones [Wooldridge, 2002]. En este punto es importante recordar que en el entorno normalmente hay también otros agentes, que trabajan en paralelo, y que por tanto pueden modificar el entorno mientras el agente deliberá, lo cual añade mayor incertidumbre al problema. Por ello, Russel y Norvig [Russell y Norvig, 2003] hacen énfasis en la idea de que un agente puede fallar, bien en la acción que decide tomar o en el momento en que decide tomarla. Es importante, por tanto, que un agente observe su entorno una vez realizada una acción para considerar si ha tenido el efecto esperado o no, y en qué medida. En este sentido, un agente debería utilizar estas observaciones para intentar mejorar sus decisiones en el futuro.

Para estructurar el componente que controla los sensores y actuadores, maneja y procesa el modelo del mundo, se han propuesto varias arquitecturas que generalmente se resumen en tres básicas: deliberativas, reactivas e híbridas.

20.2.1 Agentes deliberativos

Las arquitecturas deliberativas utilizan modelos de representación simbólica del conocimiento y suelen basarse en la teoría clásica de planificación. Los agentes tienen un conjunto de objetivos y, dependiendo del estado del mundo, generan planes para alcanzar los objetivos.

Cuando se decide implementar una arquitectura deliberativa hay que buscar, en primer lugar, una descripción simbólica adecuada del problema, e integrarla en el agente, para que éste pueda razonar y llevar a cabo las tareas encomendadas en el tiempo preestablecido. Aunque parece una cuestión trivial, debido a la complejidad de los algoritmos de manipulación simbólica, es un aspecto al que hay que prestar mucha atención, especialmente si se tiene en cuenta que los agentes se desenvuelven en entornos reales, en los que frecuentemente tienen que responder a los estímulos en tiempo real.

La arquitectura deliberativa más extendida es la denominada BDI (del inglés, *Belief, Desire, Intention*) [Rao y Georgeff, 1991]. Esta arquitectura se caracteriza por el hecho de que los agentes que la implementan están dotados cada uno de un *estado mental*. Un estado mental es un conjunto de Creencias, Deseos e Intenciones.

Las creencias representan el conocimiento que el agente tiene del entorno. Desde un punto de vista informático, son la forma de representar el estado del entorno, por ejemplo el valor de una variable, los valores de una base de datos relacional, o expresiones simbólicas del cálculo de predicados. En entornos dinámicos es necesario mantener información sobre los eventos pasados al mismo tiempo que se debe permitir su adaptación y evolución, por lo que una buena gestión de estos elementos es fundamental para construir agentes eficientes.

Los deseos son los objetivos que persigue el agente, su visión ideal del mundo. Un objetivo puede simplemente ser el valor de una variable, un registro, o una expresión simbólica en alguna lógica. Un objetivo representa un estado final deseado. El software convencional está *orientado a la tarea* en lugar de *al objetivo*, de forma que cada tarea u operación se ejecuta sin ningún recuerdo de por qué ha comenzado su ejecución. Esto significa que el sistema no puede recuperarse ante fallos automáticamente (a menos que esto sea específicamente codificado por el programador) y no puede descubrir ni aprovechar oportunidades que surjan inesperadamente. Por ejemplo, la razón de que un humano sea capaz de superar la pérdida de un tren o un pinchazo inesperado de la rueda de su coche es porque conoce dónde está (a través de sus creencias) y recuerda qué quiere conseguir (a través de sus objetivos). La semántica fundamental de los objetivos, sin tener en cuenta cómo se representa computacionalmente, se reflejaría en una lógica de los deseos.

Para alcanzar los objetivos propuestos, a partir de las creencias existentes es necesario definir un mecanismo de planificación que permita identificar las intenciones. En este sentido, los agentes están inmersos en sistemas dinámicos en los que en ocasiones será necesario decidir durante la ejecución del plan inicialmente seleccionado, ante cambios en el entorno, si se replanifica o no se replanifica. El sistema necesita comprometerse con los planes y subobjetivos, pero también ser capaz de reconsiderar éstos en los momentos clave. Estos planes vinculados a la consecución de un objetivo constituyen las intenciones del agente. Las intenciones son simplemente un conjunto de caminos de ejecución que pueden ser interrumpidos de una forma apropiada al recibir información acerca de cambios en el entorno.

Existen variaciones en la formulación teórica del modelo BDI que no se van a detallar aquí, pero el lector interesado puede encontrar una descripción bastante comprensible en el capítulo 2 de [Ana Mas, 2005].

La arquitectura del agente BDI debe contemplar entonces la gestión y el control del estado mental, tal como muestra la Figura 20.2. El gestor de estado mental proporciona los mecanismos para crear, modificar y destruir entidades del estado mental, manteniendo su consistencia. En su forma más genérica las entidades del estado mental son las creencias y los objetivos (deseos del agente). El control de estado mental encapsula el mecanismo de decisión y genera las intenciones del agente: planes o secuencias de tareas que el agente intentará ejecutar para lograr satisfacer sus objetivos. Obsérvese en la figura que los eventos del entorno, capturados por los sensores, son notificados al control de estado mental que decidirá cómo afecta a su estado mental. Este punto es importante, ya que la recepción de un evento puede tener varios efectos:

- Un evento puede ser una evidencia de que un objetivo del agente se ha cumplido o que ha fallado. Esto modificará el estado del objetivo y puede ocasionar la necesidad de replanificar.
- El evento puede ser una información que el agente considere útil guardar como un hecho sobre el mundo, y por tanto puede incorporarse al modelo de creencias.
- También puede ocurrir que el evento no tenga significado para el agente en el contexto actual y decida simplemente ignorarlo.

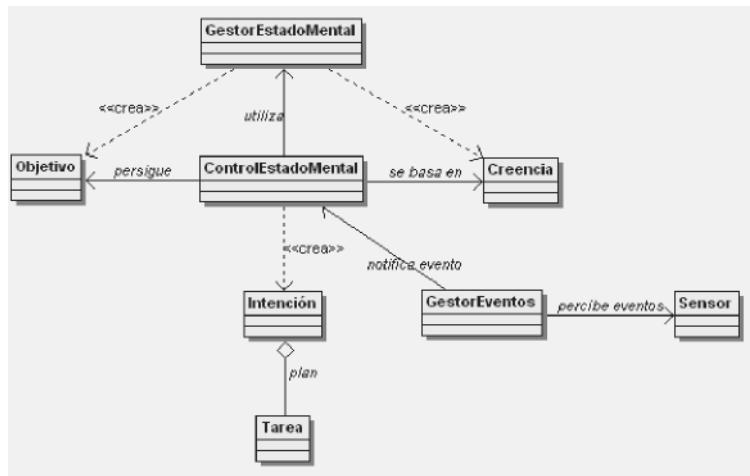


Figura 20.2: Componente de control en un agente BDI.

La implementación de esta arquitectura genérica se puede basar en múltiples mecanismos. Por ejemplo, modelando la experiencia del agente y la decisión mediante razonamiento basado en casos (CBR), como se propone en [Corchado y Laza, 2003], que se ha utilizado para implementar un sistema de ayuda a turistas en teléfonos móviles [Pavón y otros, 2004]. O bien, incluyendo lógica borrosa en el razonamiento

interno del agente para que pueda determinar el grado de confianza en otros agentes de un mercado electrónico [Carbó y otros, 2003].

20.2.2 Agentes reactivos

La utilización de una representación simbólica y mecanismos de toma de decisiones puede resultar costosa en tiempo y recursos, por lo cual algunos investigadores han considerado necesario buscar modelos más efectivos de representación del conocimiento y de comportamiento para los agentes. Esta asunción puede ser válida en situaciones donde las acciones del agente se producen como respuesta directa a los eventos del entorno, sin requerir un proceso deliberativo complejo. En estos casos, la inteligencia surgiría por la interacción de múltiples agentes, aunque cada uno tuviera un comportamiento muy sencillo.

El ejemplo más clásico de estas arquitecturas es la *arquitectura de subsunción* [Brooks, 1991], que se basa precisamente en esta hipótesis: no es necesario que los agentes tengan una representación simbólica y que puedan razonar sobre ella para que el conjunto del SMA pueda ofrecer un comportamiento inteligente. Este comportamiento inteligente puede surgir de la interacción entre los agentes.

En una arquitectura de subsunción las tareas que definen un comportamiento están organizadas en jerarquías de capas, de menor a mayor nivel de abstracción (véase la Figura 20.3).

Los comportamientos son sencillos, y se pueden implementar como autómatas simples o como reglas del tipo *situación* → *acción*, que básicamente asocian acciones a la aparición de determinados eventos de entrada.

Como en un momento dado podrían activarse varios comportamientos a la vez, para evitar acciones contradictorias, los comportamientos de los niveles inferiores pueden inhibir las entradas o salidas de los de niveles superiores. De esta manera, los niveles inferiores tratan eventos de mayor prioridad.

Un ejemplo de esta arquitectura se propuso para la realización de prospecciones en terrenos inhóspitos, como en el caso de la exploración planetaria [Steels, 1990]. Aquí, varios robots tienen que recoger un conjunto de muestras minerales cuya distribución no se conoce inicialmente, aunque se sabe que suele presentarse en cúmulos. Cada robot va recogiendo muestras y al cabo de un rato vuelve a la nave nodriza a depositar su carga. Debido a ciertas características del terreno (montículos) no es posible que los robots se comuniquen directamente por radio en todo momento. El problema es cómo se organizan para hacer la exploración de la manera más eficiente y completa.

Para ello, se proponía utilizar dos mecanismos. Por una parte, se genera un campo gradiente, mediante una señal que emite la nave nodriza y que por tanto, al alejarse de ella, será más débil. Los robots pueden saber así si se alejan o se acercan a la misma. Por otra parte, se plantea un sistema de comunicación indirecta entre los robots, mediante *migas* radiactivas que los agentes pueden dejar en el terreno, detectar y recoger al pasar. Así, si un robot deja algunas de estas migas radioactivas en algún lugar, otro robot que se acerque por el mismo lo podrá detectar.

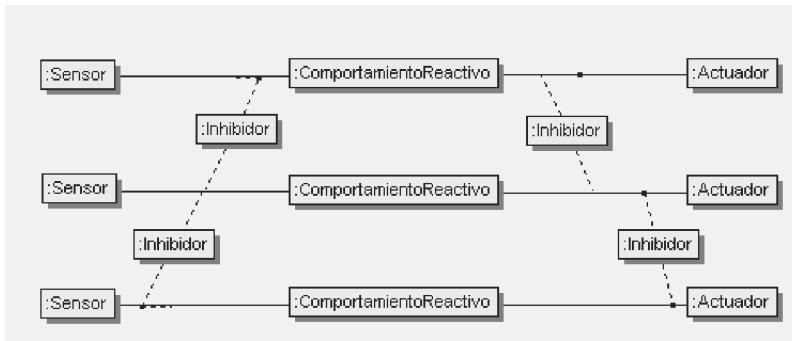


Figura 20.3: Arquitectura de subsunción.

El comportamiento de cada robot se construye a partir de varios comportamientos sencillos:

1. *Si* se detecta un obstáculo, *entonces* cambiar de dirección.
2. *Si* se llevan muestras y se está en la base, *entonces* descargar las muestras.
3. *Si* se llevan muestras y no se está en la base, *entonces* dejar dos migas y moverse hacia mayor gradiente.
4. *Si* se detecta una muestra, *entonces* recogerla.
5. *Si* se detectan migas, *entonces* recoger una y moverse hacia menor gradiente.
6. *Si* cierto, *entonces* moverse de manera aleatoria.

Obsérvese que teniendo en cuenta la prioridad de las reglas, un robot lo primero que hará siempre es evitar obstáculos. Si está en la base dejará las muestras que lleve. Cuando un robot encuentra un cúmulo de muestras, recogerá unas cuantas y en su vuelta a la base (nave nodriza) irá dejando migas por el camino. De esta manera, otros robots que las encuentren sabrán por dónde está el cúmulo y podrán ayudarle a recoger muestras del mismo. De esta manera surge la cooperación entre los robots mediante la formulación de un comportamiento muy sencillo y sin necesidad de razonamiento abstracto.

Estas arquitecturas reactivas son interesantes cuando los problemas están bien definidos. La emergencia de un comportamiento inteligente como consecuencia de la interacción de los agentes debe estar bien diseñada, como en el ejemplo anterior y hay que prever los posibles efectos laterales. Es difícil establecer una metodología para diseñar este tipo de sistemas y generalmente requieren un proceso laborioso de experimentación mediante prueba y error bastante intenso. Por esta razón, el número de capas que se pueden considerar es algo limitado y no puede ir más allá de una decena ya que las posibles interacciones entre los comportamientos y los agentes puede llegar a ser muy complejo de entender.

20.2.3 Agentes híbridos

Para superar las limitaciones de las arquitecturas puramente reactivas y deliberativas se han propuesto sistemas híbridos que pretenden combinar aspectos de ambos modelos.

Una primera propuesta puede ser construir un agente compuesto de dos subsistemas: uno deliberativo, que utilice un modelo simbólico y que genere planes en el sentido expuesto anteriormente, y otro reactivo, centrado en reaccionar ante los eventos que tengan lugar en el entorno, que no requiera un mecanismo de razonamiento complejo. Estos subsistemas se pueden utilizar para definir una estructuración en dos ejes:

- Vertical: sólo una capa tiene acceso a los sensores y actuadores.
- Horizontal: todas las capas tienen acceso a los sensores y a los actuadores.

Estas capas se organizan jerárquicamente con información sobre el entorno a diferentes niveles de abstracción. La mayoría de las arquitecturas encuentran suficientes tres niveles:

1. *Reactivo*, o de más bajo nivel. En ésta se toman decisiones acerca de qué hacer con eventos recibidos del entorno en tiempo real. Suele estar implementado mediante arquitecturas de subsunción.
2. *Conocimiento*, o nivel intermedio. Trata el conocimiento que el agente posee del medio, normalmente con la ayuda de una representación simbólica del mismo.
3. *Social*: es la capa de más alto nivel. En ella se manejan los aspectos sociales del entorno, incluyendo tanto información de otros agentes, como los deseos, intenciones, etc.

El comportamiento global del agente queda definido por la interacción entre estos niveles. Esta interacción cambia de una arquitectura a otra.

Un ejemplo de este tipo de arquitectura son las *TouringMachines* [Ferguson, 1992] (no confundir con las máquinas de Touring clásicas). Se trata de una arquitectura de tres capas horizontales (véase la Figura 20.4):

1. *Capa reactiva* (respuesta inmediata a cambios del entorno, como reglas situación→acción, o entrada de senso→salida de actuador).
2. *Capa planificadora* (comportamiento proactivo basado en esquemas de planes).
3. *Capa modeladora* (modelo del mundo para anticipar conflictos y generar nuevos objetivos para la capa planificadora).

Todas ellas están integradas en un subsistema de control basado en reglas que definen si hay que inhibir entradas y salidas, y determinan qué capa tiene control sobre el agente.

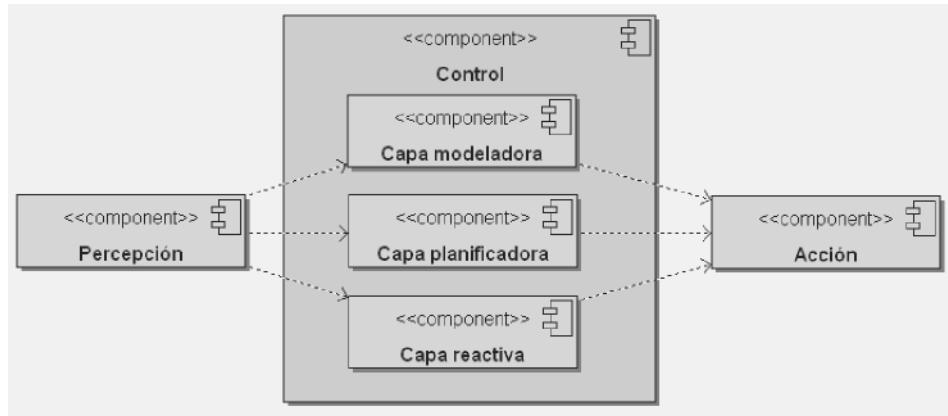


Figura 20.4: Ejemplo de arquitectura híbrida con tres capas. El componente de control puede inhibir las entradas y las salidas a cada capa.

Otra arquitectura híbrida es INTERRAP [Müller, 1996], que plantea un modo de trabajo similar pero las capas están estructuradas verticalmente, incluyendo además una base de conocimiento también organizada en capas:

1. Capa de comportamiento (reactivo): información del entorno.
2. Capa de planificación (proactivo): planes y acciones del agente.
3. Capa de cooperación (interacción social): planes y acciones de otros agentes del entorno.

Este tipo de arquitecturas aparece en muchos problemas en los que es necesario integrar un razonamiento deliberativo para la negociación conducente a una modificación de la ejecución, pero donde la ejecución es independiente del razonamiento. Un buen ejemplo son los sistemas de robótica donde se da una ejecución de acciones continua y por lo tanto el bajo nivel es reactivo y al mismo tiempo se integran los robots en un sistema multiagente para la consecución de un objetivo conjunto. El mismo problema aparece con otros tipos de sistemas reales como los sistemas de vigilancia.

En general, en un SMA suelen participar agentes de varios tipos. Algunos más simples de naturaleza reactiva que generalmente ofrecen servicios a otros más elaborados, de naturaleza híbrida o deliberativa. Depende de los objetivos del agente y la necesidad de razonar sobre su contexto que el diseñador decida aplicar un tipo de arquitectura u otro.

20.3 Sociedades de agentes

Como ya se comentó en la introducción, la sociabilidad es una característica fundamental de los agentes. Los agentes pueden colaborar o competir entre sí. Si participan

en una misma organización, los agentes cooperarán para lograr la satisfacción de objetivos comunes. Pero existen también situaciones, como en mercados de compraventa de bienes o servicios, donde los agentes pueden competir, cada uno persiguiendo maximizar sus propios beneficios. Para que todo ello pueda llevarse a cabo, han de definirse unas reglas o principios básicos que los agentes deben respetar, o mejor, que el propio sistema sea capaz de hacer cumplir.

20.3.1 Organizaciones de agentes

La organización juega un papel importante en la concepción de un SMA porque define el propósito general del sistema, su estructura, el contexto en el que los agentes desempeñan sus funciones específicas, las políticas de participación de los agentes, y las interacciones que permiten la colaboración entre los agentes.

Desde un punto de vista estructural, la organización es un conjunto de entidades asociadas por relaciones de agregación y herencia. Sobre esta estructura se definen una serie de relaciones que inducen a la formación de flujos de trabajo (dinámica de la organización) y restricciones sociales.

La estructura organizativa se define como una jerarquía de grupos, que se determinan utilizando diversos criterios. En este sentido, se pueden aplicar distintos estilos de organización ampliamente estudiados en la teoría de organización empresarial o industrial, como lineal, funcional, por comités, en divisiones o matriciales, por citar algunos ejemplos.

Un grupo se define identificando los *roles* y *recursos* que lo conforman. Un rol define una funcionalidad o las responsabilidades que tendrá que asumir el agente que lo desempeñe en un momento dado. Los recursos son utilizados por los agentes para realizar sus tareas. Asimismo, los grupos pueden estructurarse recursivamente en otros grupos, definiendo así una jerarquía. Este tipo de estructuración es útil cuando el número de entidades en el SMA es considerable. La asignación de entidades a un grupo obedece a un propósito organizacional: bien porque va a facilitar la definición de grupos de trabajo, o bien porque sus miembros tienen ciertas características comunes, como el que estén desplegados en una plataforma específica o un dominio administrativo o por una distribución funcional de los servicios del sistema.

La funcionalidad de una organización está definida por su propósito (objetivos) y las tareas que puede realizar. Una organización tiene uno o varios objetivos y depende de sus agentes para realizar las tareas necesarias para conseguir satisfacer los objetivos. Las tareas en la organización están relacionadas en flujos de trabajo donde se determinan los responsables de la ejecución de las tareas, los recursos que se les asignan y los resultados que se esperan de cada una.

En la organización también se definen *normas sociales* que especifican cómo se establecen y desarrollan las relaciones sociales entre los agentes. El respeto a estas normas puede implementarse directamente en los agentes, si se trata de un SMA cerrado donde todos los agentes están desarrollados por las mismas personas. Sin embargo, es más habitual considerar SMA abiertos, donde pueden entrar a lo largo del tiempo distintos tipos de agentes, que pueden estar implementados por terceros. En este caso es preciso que la organización defina mecanismos que aseguren el cumplimiento de las

normas sociales. Este asunto es actualmente un área de investigación importante en el ámbito de los SMA. En el ejemplo de la sección 20.7 se puede ver cómo la definición de la organización es parte integrante del diseño de un SMA.

20.3.2 Coordinación en SMA

La coordinación entre los agentes es necesaria para que puedan cooperar o colaborar en la consecución de los objetivos de una organización. La Teoría de la Coordinación [Malone y Crowston, 1994] define este término como *la gestión de dependencias entre actividades*. Por ejemplo, cómo varias tareas pueden compartir recursos o cómo modelar la relación entre distintas tareas de un flujo de trabajo con dependencias del tipo productor/consumidor. Un aspecto importante de la Teoría de la Coordinación es que las dependencias se establecen entre tareas y no entre agentes. Esto tiene la ventaja de que permite modelar más fácilmente los efectos de reasignar tareas a varios agentes, lo cual es habitual en procesos de rediseño.

La Teoría de la Coordinación identifica algunas dependencias que se encuentran normalmente en muchas organizaciones, y plantea algunos mecanismos de coordinación que suelen emplearse para su gestión [Malone y Crowston, 1994]:

Dependencia	Ejemplos de mecanismos de coordinación
Recursos compartidos	FIFO, prioridades, autoridad central, puja (Similar a recursos compartidos)
Asignación de tareas	*
Relaciones productor/consumidor	Notificaciones, seguimiento
Prerrequisitos	Gestión de inventarios
Transferencia	Estandarización, participación de usuarios
Usabilidad	Planificación, sincronización
Restricciones de simultaneidad	Selección de objetivos, descomposición de tareas
Tarea/Subtarea	

En SMA, como los agentes son autónomos y por tanto pueden decidir en tiempo de ejecución lo que hacer y no llevan las decisiones predeterminadas en su diseño, tienen que coordinar dinámicamente sus actividades. Esto es diferente de lo que ocurre normalmente en sistemas distribuidos clásicos, donde los mecanismos y estrategias de coordinación se definen claramente durante el diseño del SMA. La situación es incluso más compleja en casos donde los agentes compiten, en un mercado o en juegos, por mencionar un par de ejemplos típicos.

Existen múltiples trabajos sobre cooperación y coordinación entre agentes. Los más clásicos se pueden consultar, por ejemplo, en los dos libros ya mencionados de introducción a los SMA [Ana Mas, 2005] y [Wooldridge, 2002], que dedican capítulos específicos a este tema.

20.3.3 Negociación, confianza y reputación

La autonomía de los agentes implica que éstos no tienen por qué aceptar toda petición que se les realice. Esto hay que tenerlo en cuenta especialmente cuando los agentes son autointeresados y tienen que llegar a tomar acuerdos para intercambiar información, bienes o tareas.

Como estas situaciones son similares a las que se dan en las sociedades humanas, muchos mecanismos de negociación en SMA se inspiran en trabajos de las ciencias sociales y económicas. En concreto, la teoría de juegos, los mecanismos de mercado, subastas, la teoría del regateo, la formación de coaliciones o mecanismos de votación son comunes para resolver problemas de negociación (véase una buena descripción en [Ana Mas, 2005]).

Un aspecto bastante relacionado con este ámbito es la confianza y la reputación de los agentes. Especialmente en entornos abiertos, un ejemplo es internet, los agentes pueden ir conociendo con el tiempo nuevos agentes con los que negociar. ¿Hasta qué punto un agente se puede fiar de otro? Éste es uno de los campos actualmente en estudio dentro de la comunidad de agentes [Sabater y Sierra, 2005].

Por una parte se requiere la existencia de métricas de confianza, que midan el grado de confianza de un miembro de una comunidad virtual. Una de las propiedades que deben cumplir estas métricas es la resistencia a ataques por agentes maliciosos que quisieran abusar de la presunción de confianza. Para ello se están proponiendo distintos esquemas, algunos más centralizados con la figura de un juez que dirime sobre denuncias de algunos miembros, y otros más distribuidos, donde se espera que la propia evolución del sistema facilite la identificación de la confianza que ofrecen los agentes.

El concepto de reputación es usado habitualmente en conjunción con el de confianza. Se refiere a lo que otros opinan de un agente. Como otras cosas, la reputación se puede consultar (y comprar) en un mundo de agentes. Un agente puede solicitar a otros de su confianza información sobre la reputación de un tercero.

Una plataforma para jugar con estos conceptos es la empleada en la competición ART (que en versión española se puede encontrar actualmente en las páginas de la competición¹). Aquí se simula un mercado de arte donde participan agentes tasadores de obras de arte (cuadros) compitiendo por obtener el mayor mercado de clientes posible (y por tanto de beneficios), lo cual se logra haciendo buenas estimaciones sobre el precio de las obras de arte. Los agentes tienen diferentes grados de conocimiento sobre los cuadros según la era o periodo histórico al cual pertenecen. En este entorno un agente tasador puede pedir la opinión de otros agentes sobre un cuadro, o puede pedir a un agente su opinión sobre otros agentes (reputaciones). Al final, de lo que se trata es de valorar adecuadamente lo buenos que son otros agentes generando opiniones sobre el valor de un cuadro, pues eso permite hacer mejores tasaciones y en consecuencia recibir más clientes, y obtener más beneficios.

Estos conceptos se pueden generalizar a organizaciones, pero este punto no ha sido tratado prácticamente por ahora, y puede ser una línea de investigación novel.

20.4 Comunicación entre agentes

La consideración de los agentes como entidades intencionales aporta una nueva dimensión a la comunicación entre agentes si se compara con el paradigma de objetos. Cuando un objeto (cliente) invoca un método en otro, el objeto que recibe la

¹http://giaa.inf.uc3m.es/ART_Testbed_SPAIN/

solicitud no tiene control sobre la decisión de ejecutar dicho método: esto corresponde exclusivamente al cliente. Sin embargo, entre agentes, debido a su autonomía, dicha decisión tiene que ser acordada por ambos. Las intenciones de los dos agentes tienen que confluir para llevar a cabo el propósito de la comunicación.

En este sentido, la comunicación entre agentes se trata como acción, de la misma manera que se ha formulado en la teoría de los *actos del habla*, que parte del trabajo del filósofo John Austin [Austin, 1962]. Éste afirma que las alocuciones del lenguaje natural se pueden considerar acciones ya que pueden cambiar el estado del mundo de manera similar a las acciones físicas (un ejemplo que utiliza el autor es una declaración de guerra, cuyo impacto es desde luego muy significativo). Consideradas como acciones, se pueden identificar varios tipos de actos del habla, que se corresponden a verbos *performativos*, como por ejemplo, *pedir*, *informar*, *preguntar*, o *aceptar*. Cada acto del habla tiene tres componentes:

- La locución: el modo de producción de frases utilizando una gramática y un léxico.
- La ilocución: el acto realizado por el locutor sobre el destinatario mediante la declaración (*utterance*). Consta de una fuerza ilocutoria (F), la performativa, y un contenido proposicional (P), el objeto de la fuerza ilocutoria. Se puede representar como F(P) (o performativa(contenido)). Por ejemplo: *aserta(está nevando)* o *responde(está nevando)*.
- Perlocución: los efectos que puede tener el acto ilocutorio en el estado del destinatario y en sus acciones, creencias y juicios. Por ejemplo: *convencer*, *inspirar*, *persuadir*, *atemorizar*.

Este trabajo fue extendido por Searle [Searle, 1969] que intentó hacer una clasificación sistemática de los posibles tipos de actos del habla, proponiendo cinco clases:

- Actos asertivos: dan información sobre el mundo. Por ejemplo: *Estoy de acuerdo. 2 y 2 son 4. Estamos en clase.*
- Actos directivos: para solicitar algo al destinatario. Por ejemplo: *Siéntate. ¿Cuántas pesetas son un euro?*
- Actos de promesa: comprometen al locutor a realizar ciertas acciones en el futuro. Por ejemplo: *Mañana vuelvo a las 8. Te enviaré las fotos.*
- Actos expresivos: dan indicaciones del estado mental del locutor. Por ejemplo: *Soy feliz. Gracias. Siento lo que ha ocurrido.*
- Actos declarativos: el mero hecho de la declaración es la realización de un acto. Por ejemplo: *Estás contratado. Empezamos la clase.*

Esta teoría es la base de los lenguajes de comunicación de agentes. Dos son los más empleados, Knowledge Query and Manipulation Language (KQML), definido por DARPA [Finin et al., 1993] y FIPA ACL [FIPA, 1999], que es el estándar más ampliamente definido en la actualidad para SMA. La base de ambas propuestas es

similar, por lo cual a continuación se describe la propuesta de FIPA, que es la más extendida.

20.4.1 El lenguaje FIPA-ACL

FIPA (*Foundation for Intelligent Physical Agents*) se estableció en 1996 como un consorcio cuyo objetivo era establecer estándares para la interoperabilidad de SMA. Actualmente es un comité de estandarización del IEEE (desde 2005). La parte principal de los estándares de FIPA es precisamente el lenguaje ACL. FIPA asume que los agentes se comunican intercambiando mensajes que representan actos de habla codificados en un lenguaje de comunicación de agentes (ACL). Además, las plataformas de agentes deben proporcionar servicios adicionales que ayuden a los agentes a encontrar otros agentes (servicios de directorio) y que faciliten el transporte de mensajes.

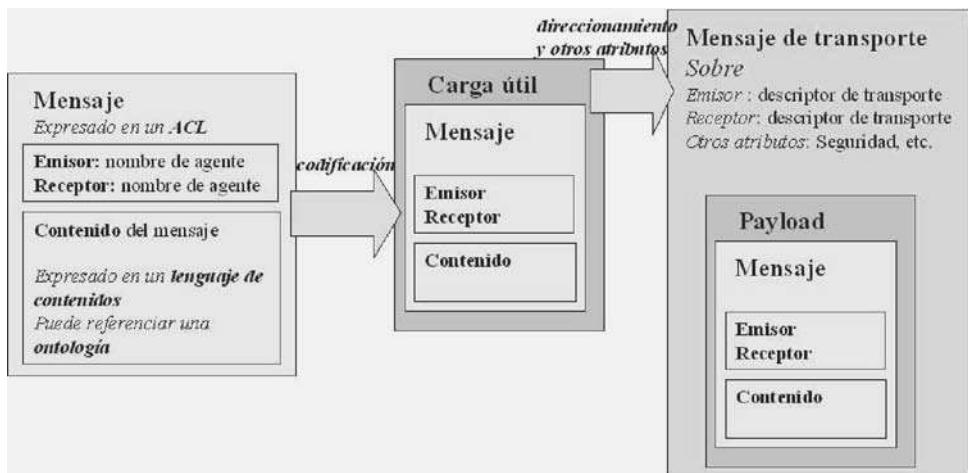


Figura 20.5: Transporte de los mensajes FIPA-ACL.

Los mensajes entre agentes son tuplas (*clave, valor*) escritos en ACL. El contenido está expresado con un lenguaje de contenidos, que no está prescrito por el estándar aunque se sugieren varias propuestas (KIF, SL, etc.). El lenguaje de contenidos normalmente hará referencia a una ontología (véase capítulo 5). Además se deben incluir los nombres del emisor y el receptor. Un mensaje puede contener recursivamente otros mensajes.

Los mensajes ACL se transportan como carga útil de un mensaje de transporte, también definidos por FIPA. La Figura 20.5 muestra la estructura del mensaje de transporte, que consta de un campo de carga útil (*payload*) y un sobre (*envelope*).

Un ejemplo de mensaje ACL sería el siguiente:

```
( inform
  :sender    AgentePujador
  :receiver   AgenteSubastador
  :content    (precio libro 1000)
  :in-reply-to ronda-4
  :reply-with puja04
  :language   sl
  :ontology   ontolibro
)
```

Aquí el AgentePujador informa al AgenteSubastador que un libro tiene un precio de 1000. La performativa, *inform* en este caso, indica la intencionalidad del agente emisor. La semántica de esta performativa se puede definir en términos del estado mental de los agentes que participan en la comunicación. Así, el agente emisor lo que pretende es cambiar el estado mental del agente receptor, para que incluya en su modelo del mundo la información que le indica en el campo de contenido.

En el mensaje ACL se añade otra información aparte de la performativa y el contenido. Hay una información del nivel de comunicaciones, que indica quiénes son el emisor y receptor del mensaje, así como información relativa al contexto del protocolo en el que se produce este mensaje (en este ejemplo se hace referencia a la cuarta ronda de una serie de pujas). También se da información del nivel de mensaje, en la que se indica el lenguaje de contenidos que se está utilizando y la ontología de los términos del contenido del mensaje.

FIPA define un conjunto amplio de performativas, que se pueden agrupar en cuatro clases principales:

- *Información*: Para indicar que una proposición (el contenido) es verdadera (*inform*), o preguntar si lo es (*inform-if*), que responderá positivamente (*confirm*) o no (*disconfirm*), o bien que no se ha entendido (*not-understood*). También se puede solicitar información sobre un conjunto de objetos que cumplan determinada proposición utilizando la performativa *inform-ref*. Y es posible suscribirse a notificaciones sobre la aparición de algún hecho utilizando la performativa *suscribe*.
- *Realización de Acciones*: Un agente solicita a otro la realización de una acción con la performativa *request*, y el receptor puede indicar que la va a realizar con *agree*, o que rehúsa la solicitud con *refuse*. En caso de que se aceptara, el solicitante podría intentar cancelar la tarea con *cancel*. Si algo fuera mal y no se pudiera ejecutar la solicitud, se puede indicar con una performativa *failure*.
- *Negociación*: Un agente puede iniciar una petición de propuestas con la performativa *cfp*, y cada propuesta recibida, con un mensaje *propose*, puede ser aceptada (*accept-proposal*) o rechazada (*reject-proposal*).

- *Intermediación*: sirve para permitir a un agente intermediario recibir solicitudes para reenviar mensajes (con las performativas *propagate* o *proxy*). Estos mensajes llevan dentro otro mensaje (el que se quiere reenviar).

20.4.2 Protocolos de comunicación

Además del lenguaje utilizado para los mensajes, FIPA define protocolos de comunicación entre agentes. Esto es bastante útil porque los protocolos se pueden ofrecer en las plataformas de agentes como componentes reutilizables estándares, facilitando así la construcción de las aplicaciones basadas en agentes.

Los protocolos definidos por FIPA son bastante habituales en conversaciones entre agentes y tienen propósitos bastante definidos:

- *Request*. Es el más habitual, para solicitar a un agente que realice cierta acción.
- *Request when*. Una variante del anterior para solicitar a un agente que realice cierta acción siempre que se cumpla una precondición.
- *Query*. Para solicitar a un agente que informe sobre algo.
- *Contract net*. Es uno de los protocolos de negociación de agentes más clásicos [Smith, 1980] y permite que un agente solicite la realización de cierta tarea a un conjunto de agentes. Éstos pueden ofertar sus propuestas basadas en unos costes y el iniciador elegirá quién la realizará finalmente.
- *English auction*. Es un tipo de subasta que se inicia con el precio más bajo y progresivamente se va subiendo.
- *Dutch auction*. Es otro tipo de subasta que se inicia con el precio más alto y progresivamente se va bajando.
- *Brokering*. Un agente (broker) ofrece las funcionalidades de otros agentes o reenvía las peticiones al agente apropiado.
- *Recruiting*. Similar al brokering, pero las respuestas sobre el servicio van directamente al agente que lo necesita (no a través del broker).
- *Propose*. El iniciador propone a una serie de agentes la realización de una tarea y éstos aceptan o no.
- *Subscribe*. Permite a un agente suscribirse para recibir notificaciones cada vez que cierta condición se vuelve verdadera (por ejemplo, la aparición de un evento).

Estos protocolos están especificados utilizando mensajes del lenguaje ACL con diagramas de secuencia que extienden los de UML para expresar alternativas y concurrencia. Un ejemplo de este tipo de diagramas es el que representa el protocolo *Contract Net*, en la Figura 20.6. Un agente inicia el protocolo con un mensaje *cfp* (petición de propuestas, en inglés, *call for proposals*), en el que especifica las características de unas tareas que desea ver realizadas. A este mensaje pueden responder

otros agentes notificando su disposición, y opcionalmente un precio, antes de una fecha límite. Posteriormente, el agente inicial podrá delegar las tareas a uno o más agentes de los que hubieran contestado positivamente, los cuales finalmente informarán del resultado de la tareas delegadas en ellos.

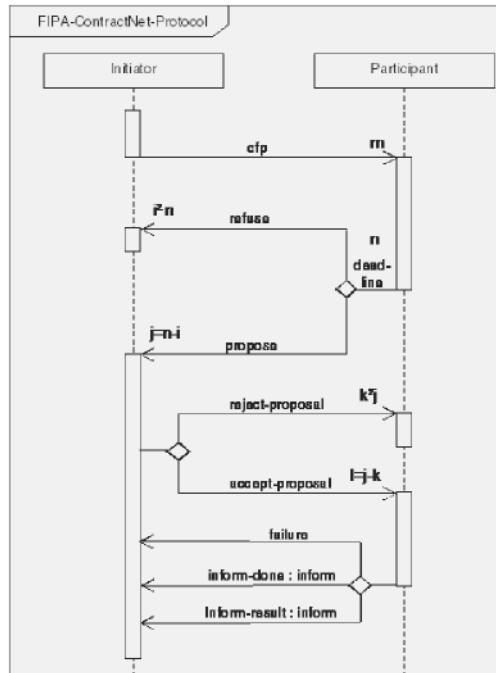


Figura 20.6: Protocolo Contract Net.

20.5 Métodos, herramientas y plataformas

La construcción de SMA integra tecnologías de distintas áreas de conocimiento: técnicas de Ingeniería del Software para estructurar el proceso de desarrollo, técnicas de IA para dotar a los programas de capacidad de adaptación y toma de decisiones y técnicas de Programación Concurrente y Distribuida para tratar la coordinación de tareas ejecutadas en diferentes máquinas. Debido a esta combinación de tecnologías, el desarrollo de SMA no es, en principio, una tarea simple. Además, los SMA generalmente tratan de dar solución a problemas que ya de por sí son complejos.

Las técnicas convencionales de ingeniería para sistemas orientados a objetos, como el Proceso Unificado, y otras, como CommonKADS, para sistemas de gestión de conocimiento, no tienen en cuenta las necesidades de especificación de los SMA, derivadas

esencialmente de su carácter distribuido, tales como la especificación de la planificación de tareas, el intercambio de información mediante lenguajes de comunicación orientados a agentes, la movilidad del código o la motivación de los componentes del sistema.

Por ello en los últimos años se están proponiendo nuevas metodologías para desarrollar software basado en los conceptos y técnicas descritas en las secciones anteriores. En estas metodologías se proporcionan notaciones para modelar SMA, métodos para análisis y diseño de SMA, y herramientas para facilitar el desarrollo.

Casi todas las metodologías de desarrollo de SMA extienden el planteamiento de la orientación a objetos con elementos que caracterizan los SMA: aspectos sociales (organizaciones, interacciones, negociación, colaboración), de comportamiento (autonomía, estado mental, objetivos, tareas), de concurrencia y de distribución. Esto tiene la ventaja de aprovechar la experiencia del campo de los objetos y facilitar la transición a los agentes, ya que la mayoría de los ingenieros software están familiarizados con ese paradigma.

Un aspecto fundamental para el desarrollo, especialmente en las actividades de análisis y diseño, es el modelado de los SMA. En general, todas las metodologías proponen considerar varios puntos de vista para tratar la complejidad de los SMA [Gómez-Sanz y Pavón, 2004]. Aunque cada metodología define los suyos propios, suele haber cierto consenso en considerar al menos los siguientes:

- Agente: ofrece la visión micro del SMA, esto es, los aspectos relacionados con los agentes individuales. Puede tratarse de la definición de la arquitectura de los agentes, de identificar las entidades que configuran su estado mental, etc.
- Organización: permite describir la visión macro del SMA. Ya se ha hablado en la sección 20.3 sobre este tema.
- Entorno: donde se sitúa y con el que interacciona el SMA.
- Interacciones: entre los agentes o de los agentes con los usuarios (generalmente humanos). Las interacciones con otras aplicaciones se considerarían parte del entorno.

Dependiendo de las características del problema, el proceso de desarrollo puede dirigirse por uno de estos puntos de vista. Si se conocen los flujos de trabajo de una organización, los agentes se pueden identificar como responsables de las tareas de la misma. Si el problema es esencialmente de cooperación de entidades distribuidas, entonces habrá que centrarse en los aspectos de coordinación e interacciones entre agentes. En robótica puede que haya que centrarse más en el entorno, que define los tipos de eventos a los que tendrán que reaccionar los agentes de control de los robots. Si se trata de hacer una simulación social, entonces habrá que centrarse en los actores, los agentes, y ver qué comportamiento emerge de las interacciones entre los mismos.

Más adelante, en la sección 20.7, se ilustra con un ejemplo cómo se podría realizar el análisis y diseño de un SMA. En este caso el desarrollo estará dirigido por los objetivos, que suele ser bastante común en SMA, teniendo en cuenta la intencionalidad de los agentes.

20.5.1 Programación de agentes

La mayoría de los SMA están programados con el lenguaje Java, esencialmente por su buena adecuación a Internet, un medio habitual para la distribución. Puesto que Java ofrece muchas bibliotecas de clases para interfaz de usuario, comunicaciones, etc., resulta relativamente cómodo añadir las clases y métodos específicamente requeridos para la implementación genérica de un SMA. Además, Java facilita la integración con otros paradigmas de programación, por ejemplo, la programación lógica (Ciao-Prolog, Prolog-Café, JavaLog, etc.) o los sistemas basados en reglas de producción (JESS o ILOG Jrules), lo que permite implementar cada agente empleando alguno de estos paradigmas y luego integrarlo en la plataforma común.

Utilizando herramientas como Ciao-Prolog o JESS, se puede dotar de cierta capacidad de razonamiento a los programas Java, mediante la incorporación de conocimiento declarativo en forma de reglas y hechos. Asimismo, la incorporación de un motor de inferencias, que ocupa relativamente poco espacio, hace que el resultado sea suficientemente eficiente. En el caso de JESS se tiene un motor de inferencias hacia adelante basado en el algoritmo Rete y es posible representar como hechos objetos computacionales Java. Esta capacidad facilita la codificación del control del agente, ya que permite codificar sus acciones mediante objetos Java que se invocarían desde las reglas JESS. De esta forma, se desacopla el control del agente de la codificación de sus actuadores. Además de ganar en reutilización, se facilita la depuración, pues muchos de los fallos de programación se localizan en el código Java más que en el código JESS. Sin embargo, también hay que comentar que trabajar con JESS requiere cierta disciplina. No sólo porque JESS no se compila, lo que limita la detección de errores semánticos, sino porque el número de reglas que pueden existir en un momento dado sea excesivo. Esta situación aparece incluso al codificar un agente reactivo de la forma más simple, como un conjunto de reglas de producción con eventos relacionados con la percepción en la parte izquierda y la invocación de los actuadores en la parte derecha. Esta codificación no es recomendable para la mayoría de agentes ya que, como en los sistemas expertos, el número de reglas se dispara enseguida. Para tratar con un número alto de reglas, el desarrollador tiene que asignar prioridades a las reglas y codificar metareglas que regulen la ejecución de otras reglas controlando los hechos que las disparan, por ejemplo. Éste es el esquema general que se sigue en la mayoría de los sistemas expertos.

Para agentes BDI, el propio modelo BDI puede utilizarse para estructurar las reglas a disparar. Esta estructuración puede realizarse de varias maneras, por ejemplo, atendiendo principalmente a cómo se gestionan los objetivos del agente. En esta gestión los objetivos siguen un ciclo de vida: se crean, se refinan (descomponen en subobjetivos), se intentan resolver (mediante una tarea) y quedan resueltos o fallidos (si se alcanzan o no las condiciones requeridas por el objetivo). Así pues, un objetivo puede recibir la siguiente secuencia de acciones:

$$\text{Activar} \rightarrow \text{Refinar} \rightarrow \text{Resolver}$$

o bien,

$$\text{Activar} \rightarrow \text{Refinar} \rightarrow \text{Resolver} \rightarrow \text{Gestionar Fallos}$$

Para ejecutar estas acciones hay que crear reglas específicas. De esta manera, el número de reglas se dispara a poco que se tengan unos pocos objetivos. Si bien el tratamiento de los objetivos puede generalizarse, no así las distintas condiciones bajo las cuales se puede considerar un objetivo resuelto. Además del incremento de reglas, hay que valorar la estructuración que se gana. Todo se estructura alrededor de los objetivos y de su ciclo de vida. Esto facilita niveles adicionales de control, por ejemplo, sobre qué objetivo se va a centrar el agente, si conviene abortar la satisfacción de un objetivo, o si se pueden crear objetivos sobre la marcha. Por ello es importante definir previamente a la implementación una arquitectura interna de agente (véase la sección 20.2) que permitirá la estructuración de este tipo de reglas.

20.5.2 Plataformas de agentes FIPA

Se han desarrollado muchas plataformas con la finalidad de permitir la interconexión y ejecución de los agentes de un SMA. Para lograr cierta interoperabilidad, la mayoría sigue el estándar FIPA, que además de especificar lenguajes de comunicación de agentes, ya presentados en la sección 20.4, define también los elementos básicos de una plataforma de agentes, incluyendo servicios de localización de agentes, entre otros.

Entre las plataformas que siguen el estándar de FIPA las dos más conocidas son FIPA Open Source (FIPA-OS), desarrollada por Nortel Networks, que implementa el lenguaje de comunicación de agentes y la plataforma de gestión de agentes FIPA, y Java Agent DEvelopment (JADE), resultado de un proyecto de investigación europeo con ese nombre y que es probablemente la más utilizada actualmente.

JADE proporciona en primer lugar un conjunto de librerías para crear agentes que puedan comunicarse entre sí utilizando el lenguaje FIPA ACL y acceder a los servicios estándar de gestión de agentes FIPA (directarios y ciclo de vida de los agentes). Además, JADE proporciona varios agentes ya implementados, que pueden ser utilizados para distintos propósitos, y que están ligados a la plataforma (por ejemplo para implementar los servicios definidos por FIPA):

- El Directory Facilitator (DF), como su nombre indica, proporciona los servicios de directorio y puede conectarse a otros agentes del mismo tipo para gestionar, si fuera preciso, una estructura jerárquica de dominios.
- El Agent Management System (AMS), uno por cada instancia de plataforma Jade en ejecución, ejerce de supervisor de acceso y uso de la plataforma. El AMS proporciona un servicio de páginas blancas y de ciclo de vida, manteniendo una lista de los identificadores de los agentes (AID) y el estado en el que están. Cada agente particular, lo primero que tendrá que hacer es registrarse en el AMS para obtener un identificador válido y quedar registrado en el sistema.
- El agente Sniffer ayuda en la depuración de las interacciones entre agentes. Este agente puede interceptar mensajes entre distintos agentes, y mostrarlos gráficamente para poder depurar sociedades de agentes, viendo el intercambio de mensajes que hay entre ellos.

- El agente Introspector permite controlar el ciclo de vida de un agente en ejecución y los mensajes que intercambia, tanto los que haya en la cola de entrada o de salida.

Dentro de cada plataforma JADE se pueden definir varios contenedores, uno para los agentes del sistema, y varios para los de las aplicaciones. Cada contenedor es una implementación de los componentes de la arquitectura FIPA necesarios para ejecutar agentes en el contexto de una máquina virtual de Java (lenguaje de implementación de Jade, aunque se podría tratar de un proceso desarrollado con otro lenguaje de programación). El contenedor se encarga de gestionar la ejecución de los agentes y sus comunicaciones con agentes que pueden estar en el mismo u otros sistemas FIPA.

Para implementar un agente en JADE primero tiene que heredar de la clase *jade.core.Agent*. Esta clase proporciona métodos para iniciar el agente en la plataforma con un nombre determinado, así como para mandar y recibir mensajes de otros agentes, desde la misma u otras plataformas conformes al estándar de FIPA.

El contenido de los mensajes se puede describir con el lenguaje SL-0. Además, se pueden definir ontologías, para enviar mensajes de acuerdo con un lenguaje y ontología particular. De manera añadida, al crear una ontología se crea automáticamente un codificador/descodificador del contenido de mensajes según esa ontología particular. De esta forma, el programador no tiene que preocuparse de cómo convertir a texto una determinada información o cómo analizar el mensaje. JADE se encarga de crear las clases JAVA con la información, y al definir la ontología se indican las clases que contienen la información que se desea enviar y recibir, para que la plataforma cree los codificadores y descodificadores apropiados. JADE también proporciona implementaciones de los protocolos estándar identificados en FIPA. Estos protocolos se definen en función de mensajes asíncronos como los descritos anteriormente. Se espera de los agentes que utilicen estos protocolos para preguntar a otros y obtener información, para negociar, para pedirles que hagan algo, etcétera. La utilización de estos protocolos permite reutilizar gran cantidad de código en el desarrollo de los agentes.

Para integrar todo esto con los distintos comportamientos que puede requerir un agente, JADE proporciona una arquitectura de agente básica. El acceso a los recursos y servicios de la plataforma se hace a través de código encapsulado en clases predefinidas de JADE, *jade.core.Behaviour*, que proporcionan el método *action()* para describir la tarea asignada a cada comportamiento. Estos comportamientos permiten enviar o recibir mensajes y organizar las actividades del agente en bloques de código que se ejecutarían de forma secuencial o no determinista, entre otros. La idea es que todo el comportamiento del agente se codifique con estas clases. Sin embargo, muchos desarrolladores optan por construir sobre estos comportamientos otras capas, como una capa JESS que permita expresar más claramente la implementación de la conducta del agente.

Actualmente JADE está disponible en múltiples plataformas, incluso para dispositivos móviles con funcionalidad J2ME-CLDC. Para estos últimos hace falta utilizar las librerías JADE LEAP, que proporcionan la mayor parte de la funcionalidad requerida por el estándar FIPA, incluyendo la movilidad de agentes. En principio, el código desarrollado para JADE vale también para JADE LEAP.

20.6 Aplicaciones de los agentes

Desde las primeras experiencias, a finales de la década de los ochenta, se ha construido una gran variedad de sistemas utilizando agentes. Éstos van desde aplicaciones sencillas, como los asistentes de correo electrónico, a sistemas de tiempo real complejos, como el control de tráfico aéreo [Ljungberg y Lucas, 1992]. Un aspecto interesante de la aplicación de la tecnología de agentes es precisamente que el concepto de agente permite modelar, de manera natural, una gama tan amplia de sistemas con requisitos tan diversos.

En [Jennings y Wooldridge, 1998] se explica precisamente en qué tipos de aplicaciones es apropiado utilizar agentes. Teniendo en cuenta que se trata de una nueva tecnología, deberían aplicarse para solucionar problemas que antes no eran automatizables, bien porque no existía una tecnología capaz de dar una solución al problema o bien porque la solución existente era demasiado costosa de aplicar. También es interesante considerar su aplicación cuando pueden mejorar considerablemente (en coste, rapidez o facilidad) el desarrollo de sistemas que podrían implementarse con tecnologías existentes.

El tipo de sistemas más simples de realizar son los *funcionales*, que producen una salida a partir de una entrada aplicando una función o algoritmo más o menos complejo. Una característica de este tipo de sistemas es que no se asume que puedan producirse cambios en el entorno durante la ejecución de una función. Los sistemas reactivos, por el contrario, mantienen una interacción continua con el entorno, el cual puede cambiar haciendo que las precondiciones válidas al principio de la ejecución de un proceso no lo sean durante la ejecución del mismo e incluso que el objetivo por el cual había comenzado su ejecución deje de tener sentido. En este tipo de sistemas se encuentran los casos difíciles: los sistemas abiertos, donde la estructura y componentes del sistema pueden cambiar dinámicamente, como es el caso de la web donde continuamente aparecen nuevas herramientas y servicios, y los sistemas complejos, donde a partir de las interacciones de múltiples componentes surgen nuevos comportamientos colectivos no predecibles.

Respecto a la mejora de la eficacia que puede suponer la adopción de un SMA, en [Bond y Gasser, 1998] se proponen tres aspectos que serían adecuados:

- Distribución inherente de datos, control, experiencia y recursos. Este caso se da cuando se consideran organizaciones donde colaboran distintos tipos de expertos. Un ejemplo sería un hospital, donde los datos sobre un paciente que manipula un médico no son los mismos que los de la enfermera. Cada individuo es responsable de un conjunto de tareas, el conocimiento especializado es diferente según se trate de una enfermera o un médico, y la responsabilidad en la gestión de recursos se reparte. A cada uno se le puede asociar un agente con su experiencia, sus responsabilidades y medios, y las relaciones entre los distintos agentes permiten definir los procesos de gestión del hospital.
- La noción de agente autónomo permite modelar de manera natural una funcionalidad específica. Por ejemplo, el asistente personal de correo electrónico como

un agente especializado en ayudar al usuario en la realización de una tarea repetitiva (como eliminar los *spam*), o los juegos en red donde cada personaje se puede representar con un agente (*bot*).

- Un sistema que tenga que coexistir con componentes heredados que deben interactuar entre sí y con nuevos componentes software. Para ello se puede recubrir el acceso a las aplicaciones utilizando agentes. De hecho, uno de los primeros problemas que se han abordado ha sido cómo los agentes pueden interactuar con aplicaciones existentes, para capturar eventos o consultar el estado de las aplicaciones y, conociendo lo que está ocurriendo, poder ofrecer nueva funcionalidad sin tener que modificar el código existente (o un mínimo para poder conectar los agentes).

Revisando sistemas basados en agentes pueden identificarse varias elementos que caracterizan su aplicación. El primero a considerar es la distribución de las entidades que componen el sistema en un entorno abierto. Un ejemplo de este tipo de sistemas son los servicios web. En un sistema de estas características hay múltiples variables que son difíciles de gestionar con un paradigma de software tradicional. En un sistema abierto pueden existir múltiples elementos, de distinto tipo, que deben ser descubiertos y con los que habrá que interactuar. Asimismo, el contexto de todas las entidades en un entorno abierto cambia continuamente, ya que hay entidades que aparecen, se modifican, se destruyen, y actúan concurrentemente. Por lo tanto hay un considerable número de comportamientos y gran cantidad de información que cambia con el tiempo. Las relaciones entre las distintas entidades son de cooperación o de competición por los recursos. En estas situaciones, las estructuras de control tradicional no son siempre válidas. Cuando se realiza una acción en el entorno no es posible determinar previamente el efecto que tendrá, sólo la intención de lo que se pretende. Todos los factores anteriores contribuyen a un indeterminismo de la dinámica del sistema, para lo cual los agentes tienen capacidades de adaptación y planificación. Los agentes pueden detectar si una tarea no se está llevando a cabo como estaba previsto, y pueden decidir cambiarla por otra tarea o colaborar con otros agentes.

Otra característica del entorno habitual en los SMA es la heterogeneidad. Actualmente vemos que cada día hay más diversidad de dispositivos, con distintas capacidades de proceso y memoria, conectados a redes fijas y móviles. Esto implica mayores grados de distribución en la gestión de las entidades del sistema, en la localización del control y en las interacciones. Los agentes asumen estas consideraciones desde su fundamento, ya que se conciben como entidades autónomas que residen en un nodo de una red e incluso, en el caso de agentes móviles, pueden migrar a otros nodos de la red durante su existencia.

También desde el punto de vista del procesamiento y gestión del conocimiento, se necesitan nuevos mecanismos para procesar la información, y la interacción entre los componentes del sistema puede requerir grados de abstracción mayores, con más soporte para el tratamiento semántico de la información. En este sentido, el uso de las ontologías y los lenguajes de comunicación de agentes supone un avance respecto a los métodos de comunicación tradicionales, como los utilizados en proceso de objetos distribuidos, que están limitados a la interoperabilidad sintáctica.

Por último, hay que mencionar los aspectos, cada día más considerados, de usabilidad del software. Un ejemplo de este tipo de cuestiones es la personalización. La personalización permite que un mismo servicio se pueda ofrecer simultáneamente de manera distinta según las características de cada usuario. Ello implica sistemas altamente configurables, donde hay que realizar un proceso especial para cada usuario. Este problema se suele atacar normalmente definiendo un agente (asistente personal) para cada usuario, con capacidad para aprender y adaptarse a los cambios en el perfil del usuario.

En resumen, para decidir la conveniencia de realizar una aplicación como un SMA habrá que preguntarse si tiene características que hagan difícil su concepción con paradigmas tradicionales. Las cuestiones que habrá que plantearse son del siguiente tipo:

- ¿Se trata de un sistema distribuido abierto? ¿Pueden incorporarse dinámicamente nuevos tipos de entidades en el sistema? ¿Pueden cambiar las existentes?
- ¿Es necesario considerar una evolución del comportamiento independiente para cada uno de los componentes del sistema o para una parte significativa?
- ¿Hay incertidumbre? ¿Es posible para una entidad del sistema conocer su contexto suficientemente para poder decidir con certeza el efecto de las acciones que puede realizar?
- ¿Hace falta definir una organización de entidades que interactúan para resolver conjuntamente problemas globales? Las características sociales de los agentes permiten modelar muchos aspectos organizativos.
- ¿Hay interdependencias entre los elementos del sistema de las que pueden surgir comportamientos complejos, usualmente no predecibles?

Si la respuesta a varias de estas preguntas es positiva, entonces será conveniente concebir el sistema como una sociedad de agentes.

Una revisión reciente del estado del arte y de las aplicaciones de la tecnología de agentes, desde la perspectiva de AgentLink, una red europea de grupos de investigación en el área de agentes, ha sido publicada en [Luck y otros, 2004]. Y con más detalle, en el último capítulo del libro [Ana Mas, 2005] se describen varias aplicaciones realizadas por investigadores del área en España.

20.7 Ejercicios resueltos: Análisis y diseño de un SMA

Como ejemplo de cómo realizar el análisis y diseño de un SMA a continuación se considera la realización de un sistema de recomendación. Existen esencialmente dos estrategias para determinar si un determinado objeto (un documento, una película, una canción, un libro, un artículo de compra, etc.) es de interés para una persona. Una de ellas, consiste en comparar el objeto en cuestión con otros que el usuario haya evaluado previamente, directa o indirectamente (por ejemplo, porque lo haya

especificado explícitamente en una encuesta, o bien si ha comprado algo se supone que le gustará). Este tipo de recomendación se basa en analizar la similitud de los objetos dependiendo de ciertas características que permitan compararlos. La técnica se denomina *filtrado por contenidos*. Aunque puede ser útil en determinados casos, suele ser poco fiable cuando en la valoración del objeto hay bastante subjetividad, como en algunos de los ejemplos citados (películas, canciones). Una estrategia más eficaz consiste en definir perfiles de usuarios y agrupar en comunidades aquellos que tengan gustos similares. Si algo le gusta a un número razonable de sujetos de la comunidad, las probabilidades de que les guste al resto son similares. Esto se denomina *filtrado colaborativo*, y en este ejemplo se va a mostrar cómo se podría realizar con agentes. Para ilustrar el proceso se utiliza la metodología INGENIAS [Pavón y otros, 2005], que incorpora un conjunto de herramientas para editar especificaciones gráficas de SMA y generar código sobre distintas plataformas². Las figuras que aparecen a continuación han sido realizadas con esta herramienta.

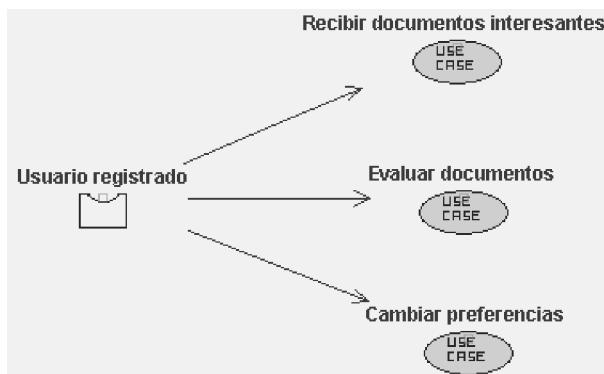


Figura 20.7: Diagrama de casos de uso para un sistema de filtrado colaborativo.

Generalmente se parte de una especificación de requisitos, utilizando por ejemplo casos de uso, como es habitual en ingeniería del software. En la Figura 20.7 se indica que un usuario registrado en el sistema lo utiliza para recibir documentos de interés y colaborará con el sistema para evaluar algunos documentos que se le propongan. También podrá cambiar su perfil de usuario. Normalmente habría que indicar otros casos de uso en el sistema real, como la autenticación de usuarios y otros servicios que proporcionara el sistema, pero para mantener el ejemplo sencillo se ha llevado a cabo cierta simplificación.

Los casos de uso pueden asociarse a objetivos del sistema, definen el propósito de la organización que se implemente como SMA. En la Figura 20.8 se muestra que estos objetivos de la organización se corresponden directamente con los casos de uso: el propósito del SMA es proporcionar la funcionalidad deseada por los usuarios.

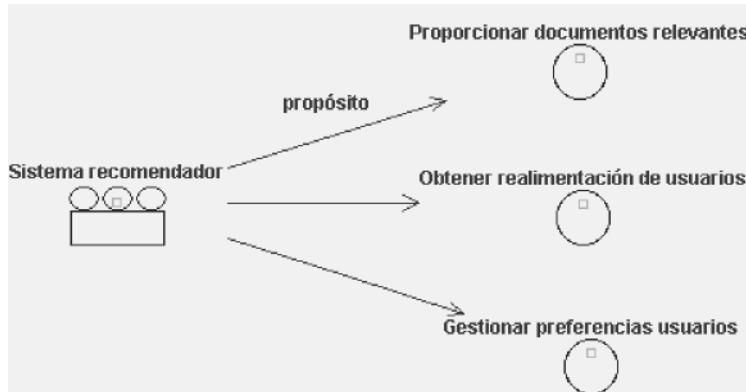


Figura 20.8: Propósito del SMA.

Estos objetivos pueden descomponerse en otros más sencillos, como parte del análisis del SMA. En la Figura 20.9 se considera uno de los objetivos de la organización, *Proporcionar documentos relevantes*, cuya satisfacción puede plantearse en términos de otros más sencillos. En este caso, para poder proporcionar documentos relevantes a los usuarios, hará falta primero conseguir documentos, y hay varias fuentes para ello, repositorios de documentos, o documentos que proporcionen los propios usuarios. Esto se refleja en el árbol de descomposición de objetivos de la figura. Asimismo, hará falta poder evaluar los documentos encontrados y ordenarlos por relevancia para poder determinar cuáles son los relevantes para los usuarios de una comunidad de usuarios de intereses similares.

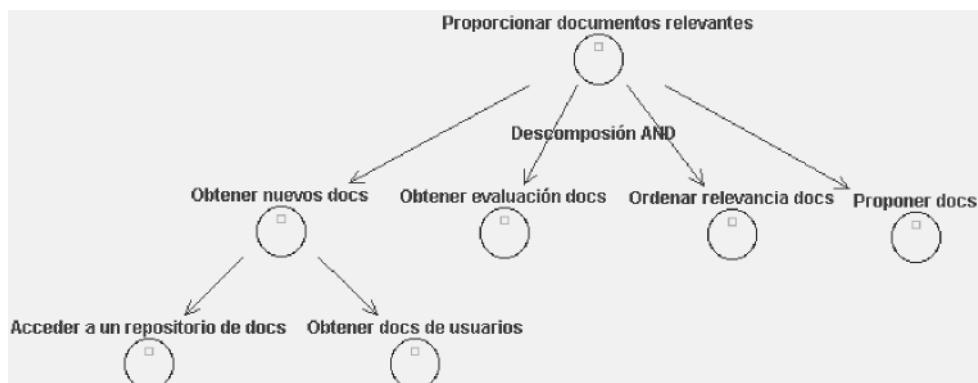


Figura 20.9: Árbol de objetivos.

²<http://ingenias.sourceforge.net>

En un momento dado, los objetivos son suficientemente concretos para que se puedan identificar tareas que contribuyan a su consecución. Inicialmente se pueden identificar simplemente las tareas, como en la Figura 20.10 para el objetivo *Obtener evaluación docs*. Posteriormente, habrá que definir las relaciones de precedencia entre las mismas, qué recursos necesitan utilizar y qué producen, en el contexto de un plan o flujos de trabajo, como aparece en la Figura 20.11. En el caso de SMA es habitual hablar de flujos de trabajo (en inglés, *workflows*) porque está más relacionado con la concepción organizacional del mismo.

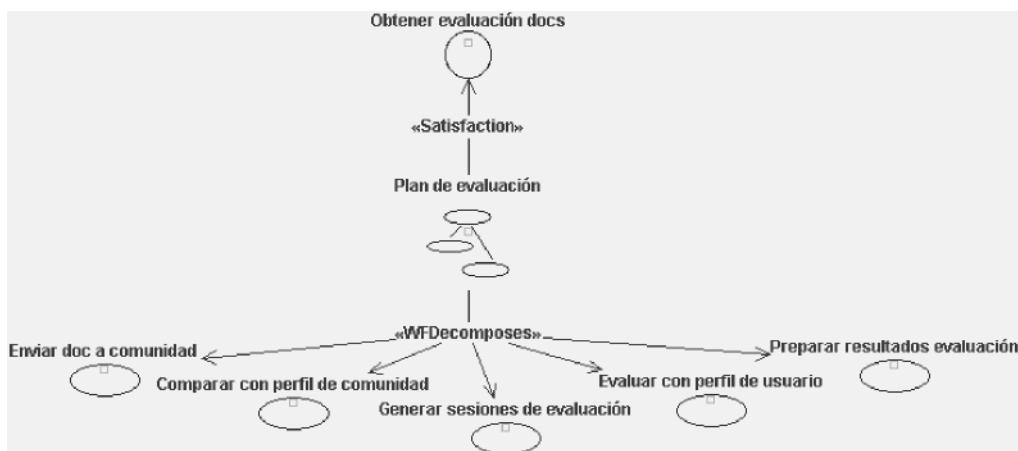


Figura 20.10: Relación entre objetivos y tareas: plan para evaluar documentos.

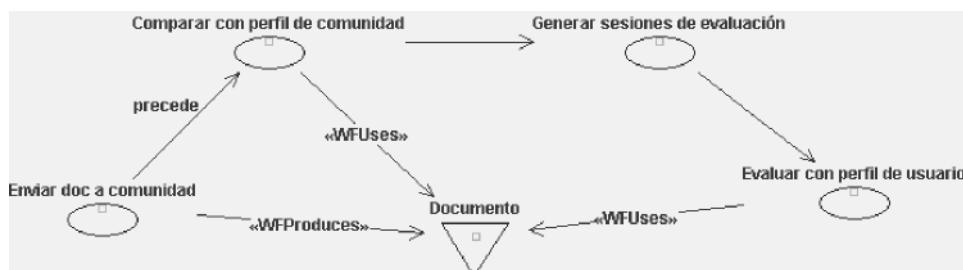


Figura 20.11: Relaciones entre tareas de un plan o flujo de trabajo.

Las tareas estarán asignadas a distintos responsables dentro del SMA. Desde un punto de vista organizativo, estas responsabilidades se asignan a roles. En la Figura 20.12 se muestran los roles *Comunidad*, responsable de tres tareas, y *Miembro*, responsable de la tarea *Evaluar con perfil de usuario*.

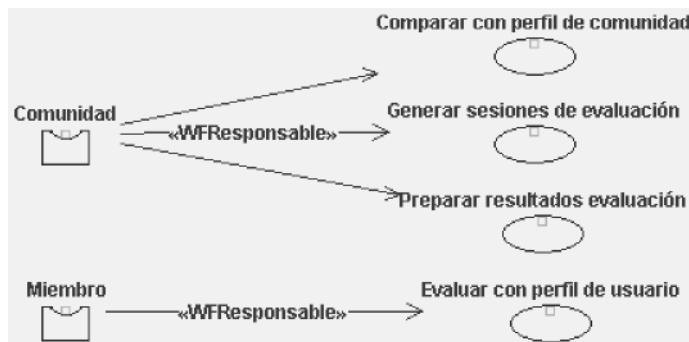


Figura 20.12: Asignación de tareas a roles de la organización.

Para colaborar, los agentes tienen que solicitar información o la realización de tareas a otros. Las relaciones de unas tareas y otras en el flujo de trabajo permite identificar qué tipos de interacciones son necesarias en el SMA. Estas interacciones se definen normalmente como un intercambio de mensajes entre agentes, aunque se podrían utilizar otros mecanismos. En el ejemplo de la Figura 20.13 se describe una interacción donde un agente, desempeñando el rol de *Comunidad*, puede solicitar a otro, con el rol de *Miembro*, que evalúe un documento. Para ello, le envía un mensaje ACL, con performativa *request*, en el que el contenido es la petición de evaluar un documento y la referencia al mismo. Este mensaje se hace en el contexto de la tarea *Generar sesiones de evaluación*. El agente que desempeñe el rol de *Miembro* ejecutará la tarea *Evaluar con perfil de usuario* y pasará el resultado con un mensaje ACL, con performativa *inform*.

Una visión más completa muestra, como en la Figura 20.14 la estructura organizativa del SMA. La organización se puede estructurar en varios grupos atendiendo a la funcionalidad de que se harán responsables. En este caso se definen grupos de interés que reflejan cómo los usuarios del sistema se pueden asociar por tener intereses similares. En cada grupo se identifican roles. Cada rol puede llevar asociado un objetivo que describe su propósito en la organización. Dentro de estos grupos hay un rol *Comunidad*, que representa al grupo de interés en su conjunto, y el rol *Miembro*, que representa cada usuario participante en el mismo.

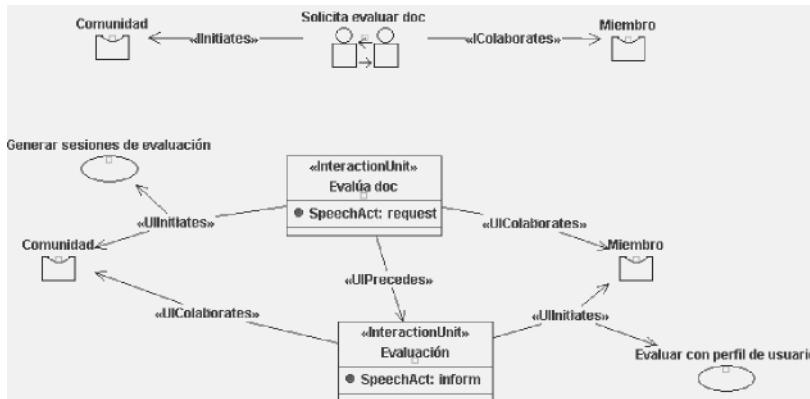


Figura 20.13: Diagrama de interacción para solicitar la realización de una evaluación.

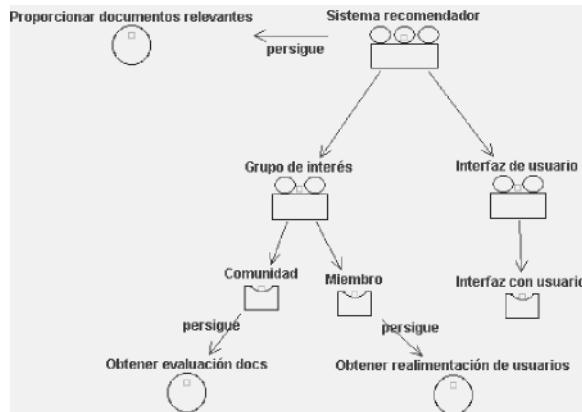


Figura 20.14: Estructura organizativa del SMA.

Obsérvese que todavía no se ha hablado de agentes, ya que durante el diseño de la organización lo habitual es identificar los puestos funcionales de la misma. ¿Quiénes ocuparán estos puestos?, esto es, ¿quiénes desempeñarán estos roles? (los agentes), se determinará más adelante y podrían, incluso, reasignarse dinámicamente. En este ejemplo una posible asignación vendría derivada de la definición de dos agentes para este sistema, uno representando a la Comunidad y otro al usuario, el *agente personal*, que se encarga también de gestionar la interfaz con el usuario, tal y como se detalla en la Figura 20.15.

Una vez identificados los agentes y los roles que pueden desempeñar en la organización, sería posible pasar a la implementación de los mismos. Bien se pueden programar directamente para ejecutar en una de las plataformas de agentes, tal como se describe en la sección 20.5, o bien se pueden utilizar algunas facilidades de generación de código que ofrezcan las herramientas. En el caso del kit de desarrollo de

INGENIAS se podría invocar al generador de código para JADE y éste iría indicando la información que falta por completar en la especificación (aquí sólo se ha mostrado una parte de los diagramas, pero habría que completarlos para definir, por ejemplo, todas las interacciones) para poder realizar la generación de código.

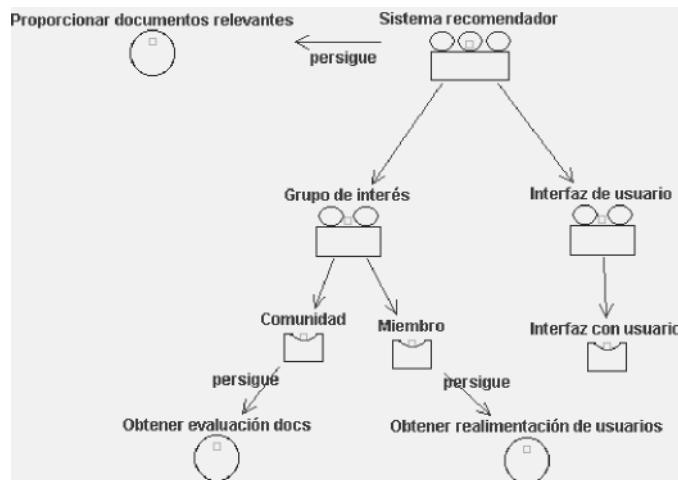


Figura 20.15: Definición de los roles que desempeñan los agentes del sistema.

20.8 Ejercicios propuestos

- 20.1.** Instalar la plataforma JADE, que se puede descargar libremente de *jade.tilab.com* y tratar de ejecutar alguno de los ejemplos del tutorial.
- 20.2.** Utilizando JADE, implementar los agentes del ejemplo de la sección 20.7.
- 20.3.** Para probar el sistema, definir agentes que simulen el comportamiento de usuarios.
- 20.4.** Si hubiera usuarios maliciosos que introdujeran documentos de poco interés al resto, por ejemplo, publicidad, ¿cómo se podría lograr que el sistema los expulsara para que no perturbaran el normal comportamiento de las comunidades virtuales?
- 20.5.** Leer la información y descargar la plataforma ART³ para realizar varios agentes, con estrategias diferentes, que compitan en un mercado de arte.

³http://giaa.inf.uc3m.es/ART_Testbed_SPAIN/el_testbed.htm

Referencias

- ANA MAS: *Agentes Software y Sistemas MultiAgente: Conceptos, Arquitecturas y Aplicaciones*. Pearson-Prentice Hall, Madrid, 2005.
- AUSTIN, JOHN: *How To Do Things With Words*. Oxford University Press, Oxford, 1962.
- BOND, A.H. y GASSER, L.: «An Analysis of Problems and Research in DAI.» *Readings in Distributed Intelligence.*, 1998, pp. 3–35.
- BROOKS, RODNEY A.: «Intelligence without Representation.» *Artif. Intell.*, 1991, **47(1-3)**, pp. 139–159.
- CARBÓ, JAVIER; MOLINA, JOSÉ M. y DÁVILA, J.: «Trust Management through Fuzzy Reputation». *Int. J. Cooperative Information Systems*, 2003, **12(1)**, pp. 135–155.
- CORCHADO, JUAN M. y LAZA, ROSALIA: «Constructing deliberative agents with case-based reasoning technology.» *Int. J. Intell. Syst.*, 2003, **18(12)**, pp. 1227–1241.
- FERGUSON, INNES A.: «Touring Machines: Autonomous Agents with Attitudes.» *IEEE Computer*, 1992, **25(5)**, pp. 51–55.
- FININ ET AL., T.: *Specification of the KQML Agent Communication Language*. DARPA Knowledge Sharing Initiative External Interfaces Working Group, Estados Unidos, 1993.
- FIPA: *Agent Communication Language*. <http://www.fipa.org>, 1999.
- GÓMEZ-SANZ, JORGE J. y PAVÓN, JUAN: «Methodologies for Developing Multi-Agent Systems.» *J. UCS*, 2004, **10(4)**, pp. 359–374.
- JENNINGS, N. y WOOLDRIDGE, M.: «Applications of Intelligent Agents.» *Agent Technology: Foundations, Applications and Markets.*, 1998, pp. 3–28.
- LJUNGBERG, M. y LUCAS, A.: «The Oasis Air Traffic Management System.» En: *Proc. of the Second Pacific Rim Int. Conference on Artificial Intelligence (PRI-CAI'92)*, , 1992.
- LUCK, MICHAEL; McBURNEY, PETER y PREIST, CHRIS: «A Manifesto for Agent Technology: Towards Next Generation Computing.» *Autonomous Agents and Multi-Agent Systems*, 2004, **9(3)**, pp. 203–252.
- MALONE, THOMAS W. y CROWSTON, KEVIN: «The Interdisciplinary Study of Coordination.» *ACM Comput. Surv.*, 1994, **26(1)**, pp. 87–119.
- MÜLLER, JÖRG P.: *The Design of Intelligent Agents - A Layered Approach*. volumen 1177 de *Lecture Notes in Computer Science*. Springer, 1996. ISBN 3-540-62003-6.

- PAVÓN, JUAN; CORCHADO, JUAN M.; GÓMEZ-SANZ, JORGE J. y OSSA, LUIS F. CASTILLO: «Mobile Tourist Guide Services with Software Agents.» En: *Mobility Aware Technologies and Applications, First International Workshop, MATA 2004, Floripa-nópolis, Brazil, October 20-22, 2004, Proceedings*, volumen 3284 de *Lecture Notes in Computer Science*, pp. 322–330. Springer, 2004.
- PAVÓN, JUAN; GÓMEZ-SANZ, JORGE J. y FUENTES, RUBÉN: «The INGENIAS Methodology and Tools.» *Agent-Oriented Methodologies.*, 2005, pp. 236–276.
- RAO, A. S. y GEORGEFF, M. P.: «Modeling Rational Agents within a BDI-Architecture». En: *Second International Conference on Principles of Knowledge Representation and Reasoning (KR91)*, pp. 473–484, 1991.
- RUSSELL, S. y NORVIG, P.: *Inteligencia Artificial. Un Enfoque Moderno*. Prentice Hall, México, segunda^a edición, 2003.
- SABATER, J. y SIERRA, C.: «Review on Computational Trust and Reputation Models». *Artificial Intelligence Review*, 2005, **24**(1), pp. 33–60.
- SEARLE, JOHN R.: *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
- SMITH, R.G.: «The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver.» *IEEE Trans. Comput.*, 1980, **C-29**, pp. 1104–1113.
- STEELS, L.: «Cooperation between distributed agentes through self organization.» En: *Decentralized AI – Proc. of the 1st European Workshop on Modelling Autonomous Agents in a Multi-Agent WOrkld (MAAMAW-89)*, pp. 175–196. Elsevier, 1990.
- WEISS ET AL., G.: *Proceedings of the European Agent Systems Summer School*. University of Utrecht, Holanda, 1999.
- WOOLDRIDGE, M.: *An Introduction to MultiAgent Systems*. Wiley, Reino Unido, 2002.
- WOOLDRIDGE, M. y JENNINGS, N.R.: «Intelligent Agents: Theory and Practice». *The Knowledge Engineering Review*, 1995, **10**(2), pp. 115–152.

Capítulo 21

Verificación y validación de sistemas inteligentes

Vicente Moret Bonillo y Eduardo Mosqueira Rey
Universidad de A Coruña

21.1 Introducción

¿Cómo podemos asegurarnos de que un sistema inteligente está actuando de forma similar a como lo haría un experto humano? La respuesta a esta pregunta la encontramos en el proceso de verificación y validación de sistemas inteligentes, también conocido como proceso V&V.

Los términos verificación y validación son ampliamente usados en la bibliografía y han aparecido múltiples definiciones de los mismos. Sin embargo, lo más importante que podemos identificar en todas estas definiciones es que el principal objetivo de ambos procesos es asegurar que los sistemas inteligentes ofrecen la respuesta correcta, de la forma correcta, cuando se les plantea un problema determinado [González y Dankel, 1993]. Los procesos de verificación y la validación de un sistema inteligente permiten:

1. Asegurar la calidad del producto desarrollado. De forma que todo sistema inteligente que llegue a sus últimas fases de desarrollo cumpla unos estándares de calidad. Para ello es necesario que en la propia metodología de construcción de sistemas expertos se incluya una fase de V&V [López y otros, 1990].
2. Asegurar su utilización en dominios críticos. Existen dominios en los cuales las decisiones que se tomen son muy importantes ya que, debido a sus consecuencias, no nos es posible reconsiderarlas. Estos dominios se denominan “dominios críticos”, y la aceptabilidad de un sistema inteligente que opere en tales dominios depende, en gran medida, de la realización de una fase formal de validación.

3. Asegurar su aceptabilidad en la rutina diaria. Un sistema sólo será aceptado dentro de una rutina diaria si cumple las expectativas para las que fue construido, y no comete errores. Si el sistema no tiene una fiabilidad adecuada es probable que los usuarios nunca lleguen a utilizarlo (y además harían bien) [O'Leary, 1993].

Pero la validación de sistemas inteligentes no constituye un campo de investigación bien estructurado. Se han desarrollado muchas aproximaciones *ad hoc* al problema de la validación, pero no existe una visión integral del mismo. Tampoco existe una clasificación global de los problemas de validación, ni existe una relación clara entre estos problemas y las técnicas destinadas a solucionarlos. [Gupta, 1993] señala que, entre los principales problemas existentes en la validación, cabe destacar: la falta de métricas de evaluación prácticas y rigurosas; la falta de especificaciones, lo que conduce a evaluaciones subjetivas; y la falta de herramientas de validación.

Los actuales avances en el campo de las metodologías de construcción de sistemas inteligentes permiten el desarrollo de este tipo de sistemas de una forma más eficiente y, con la inclusión de una fase de V&V en dichas metodologías, se podrá comprobar más formalmente la calidad del producto construido.

Un sistema inteligente es, al mismo tiempo, un software convencional y un modelo del conocimiento humano. La verificación y la validación de la parte software puede ser realizada siguiendo la metodología de la ingeniería del software, pero la parte propiamente heurística del sistema necesita técnicas particulares.

A pesar de que en la ingeniería del software términos como validación o verificación están bastante bien definidos, al intentar adaptar estos mismos términos a la ingeniería de conocimiento el consenso encontrado no es tan grande. Así, a pesar de que se han hecho intentos para unificar la terminología [Hoppe y Meseguer, 1993], lo normal es que cada autor desarrolle su propia definición de verificación y validación.

Aunque la V&V constituye la primera parte del análisis de comportamiento de un sistema inteligente, existen también fases posteriores. Así, el análisis de comportamiento puede verse como una pirámide basada en la verificación y validación, a partir de las cuales se desarrollan una serie de actividades que permiten asegurar la calidad del sistema, agrupadas aquí bajo el término evaluación (véase la Figura 21.1).

La *verificación* es, según [Boehm, 1981], la comprobación de que estamos construyendo el producto correctamente. A la hora de tratar sistemas inteligentes esta definición se formula de forma distinta, como por ejemplo “comprobar que el sistema no tiene errores y cumple sus especificaciones iniciales”.

La *validación*, también según [Boehm, 1981], es la comprobación de que estamos construyendo el producto correcto. Lo que, expresado en términos de sistemas inteligentes significa “comprobar que la salida del sistema es la correcta y que se cumplen las necesidades y los requisitos del usuario”.

La *evaluación* se define aquí como una fase genérica que engloba aquellos aspectos que van más allá de la corrección de las soluciones finales (como utilidad, robustez, velocidad, eficiencia, posibilidades de ampliación, o facilidad de manejo).

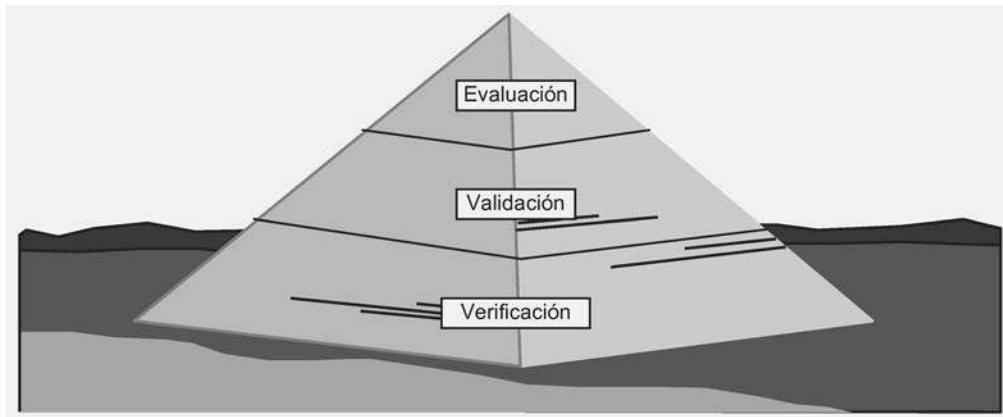


Figura 21.1: Pirámide de análisis del comportamiento de un sistema inteligente.

21.2 Verificación de sistemas inteligentes

Con la verificación se pretende comprobar que el sistema desarrollado cumple sus especificaciones de diseño, y que el software no contiene errores. La verificación es la fase más estudiada en el ámbito del análisis del comportamiento de los sistemas inteligentes, y en particular, de los sistemas basados en reglas, en los que nos centraremos en este capítulo.

Considerando los elementos y características diferenciales de los sistemas inteligentes, el proceso de verificación incluye las siguientes tareas: (a) verificación del cumplimiento de las especificaciones, (b) verificación de los mecanismos de inferencia, y (c) verificación de la base de conocimientos.

21.2.1 Cumplimiento de las especificaciones

El análisis de las especificaciones puede ser llevado a cabo por los desarrolladores, por los usuarios, por los expertos, o por un grupo de evaluadores independientes. Según [González y Dankel, 1993], las cuestiones más importantes que se deben abordar en este proceso consisten en comprobar si:

- Se ha implementado el paradigma de representación del conocimiento adecuado.
- Se ha empleado la técnica de razonamiento adecuada.
- El diseño y la implementación del sistema han sido llevados a cabo modularmente.
- La conexión con el software externo se realiza de forma adecuada.
- El interfaz de usuario cumple las especificaciones.

- Las facilidades de explicación son apropiadas para los potenciales usuarios del sistema.
- Se cumplen los requisitos de rendimiento en tiempo real.
- El mantenimiento del sistema es posible.
- El sistema cumple las especificaciones de seguridad.
- La base de conocimientos está protegida ante modificaciones realizadas por personal no autorizado.

Por otra parte, la comprobación de errores en el sistema inteligente debe referirse a cada uno de sus componentes principales: los mecanismos de inferencia y la base de conocimientos.

21.2.2 Verificación de los mecanismos de inferencia

El uso de herramientas comerciales para el desarrollo de sistemas inteligentes ha reducido la dificultad de la verificación de los mecanismos de inferencia, ya que se asume que ésta ha sido realizada por los desarrolladores de la herramienta. Por ello, la responsabilidad del ingeniero del conocimiento recae fundamentalmente en la elección de la herramienta apropiada.

Sin embargo, esta asunción de “correcto funcionamiento” no siempre es cierta (sobre todo en versiones nuevas de la herramienta). Por ello, para aplicaciones que trabajan en dominios críticos, el funcionamiento correcto del sistema inteligente debe verificarse a través de distintas pruebas.

Muchas veces los problemas que surgen con las herramientas comerciales pueden no ser causa de errores en su programación. Así, en ocasiones, hay que pensar en un desconocimiento del funcionamiento exacto de la herramienta. Por ejemplo, los procedimientos de resolución de conflictos, o los mecanismos de herencia, pueden hacer difícil el seguimiento del curso exacto de la inferencia. De esta forma, aunque el conocimiento estático esté verificado, el funcionamiento final del sistema puede no ser el apropiado.

En el supuesto de que decidamos construir nuestros propios mecanismos de inferencia, será preciso realizar su verificación. Como estamos tratando software convencional (ya que un motor de inferencias es básicamente un programa convencional que incluye un intérprete y unas estructuras de control), podemos aplicar para su verificación las técnicas clásicas de la ingeniería del software.

En cualquier caso se recomienda la utilización, siempre que sea posible, de mecanismos de inferencia certificados, es decir, aquéllos cuyo funcionamiento correcto haya sido debidamente contrastado. Además, en caso de utilizar herramientas comerciales, se aconseja realizar pruebas para comprobar que efectivamente tales herramientas se comportan como se indica en sus manuales.

21.2.3 Verificación de la base de conocimientos

La verificación de la base de conocimientos es plena responsabilidad del ingeniero del conocimiento. Esta verificación se basa, generalmente, en el concepto de anomalías. Una anomalía es un uso poco común del esquema de representación del conocimiento, que puede ser considerado como un error potencial. Existen anomalías que no constituyen errores, y viceversa.

La verificación de la base de conocimientos no nos asegura que las respuestas de nuestro sistema sean correctas, lo que nos asegura es que el sistema ha sido diseñado e implementado de forma correcta.

Los aspectos que se suelen examinar a la hora de verificar una base de conocimientos son la consistencia y la completitud. A continuación, comentaremos una serie de pruebas que se realizan para comprobar que la base de conocimientos es consistente y completa. En principio, supondremos que los sistemas no manejan incertidumbre, luego veremos cómo la inclusión de incertidumbre influye sobre las pruebas practicadas.

21.2.3.1 Verificación de la consistencia

La verificación de la consistencia de la base de conocimientos permite la detección de reglas redundantes, reglas conflictivas, reglas englobadas en otras, reglas circulares, y condiciones IF innecesarias.

1. *Reglas redundantes.* Existen dos tipos de redundancias: las redundancias sintácticas y las redundancias semánticas. Las redundancias sintácticas ocurren cuando dos reglas tienen las mismas premisas y alcanzan idénticas conclusiones:

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) \\ q(x) \wedge p(x) &\rightarrow r(x) \end{aligned}$$

Por otro lado, las redundancias semánticas ocurren cuando las premisas o las conclusiones de una regla no son idénticas en la sintaxis, pero sí en el significado.

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) = \text{Tormenta} \\ q(x) \wedge p(x) &\rightarrow r'(x) = \text{Actividad Eléctrica} \end{aligned}$$

Las redundancias semánticas son menos comunes que las sintácticas, pero también son más difíciles de detectar. Las redundancias no causan necesariamente problemas lógicos, aunque pueden afectar a la eficiencia del sistema. Sin embargo, los problemas pueden aparecer cuando, en futuras versiones del sistema, se modifica una regla pero no la otra.

2. *Reglas conflictivas.* Dos reglas son conflictivas cuando sus premisas son idénticas pero sus conclusiones son contradictorias.

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) \\ p(x) \wedge q(x) &\rightarrow \neg r(x) \end{aligned}$$

No siempre que las premisas de dos reglas sean idénticas podemos decir que ha ocurrido un conflicto. Así, por ejemplo, si la conclusión es un atributo multi-valuado, podemos estar tratando de establecer la probabilidad de aparición de las distintas hipótesis. También puede suceder que el atributo esté tomado, a la vez, varios valores (por ejemplo, una persona puede ser alérgica a varias sustancias o haber sido infectada por varios organismos).

3. *Reglas englobadas en otras.* Una regla está englobada en otra si las dos reglas tienen las mismas conclusiones, pero una de ellas tiene restricciones adicionales en la premisa.

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) \\ p(x) &\rightarrow r(x) \end{aligned}$$

En este caso la regla con más restricciones en la premisa está englobada dentro de la que tiene menos. En las estrategias de resolución de conflictos entre las reglas se podría intentar ejecutar primero la regla más específica, y en caso de no ser posible, ejecutar la regla más general.

4. *Reglas circulares.* Un conjunto de reglas es circular si el encadenamiento de las mismas forma un ciclo, es decir, se comienza por una determinada condición y al final del razonamiento volvemos de nuevo a la misma condición.

$$\begin{aligned} p(x) &\rightarrow q(x) \\ q(x) &\rightarrow r(x) \\ r(x) &\rightarrow p(x) \end{aligned}$$

Los conjuntos de reglas circulares pueden provocar que el sistema caiga en un bucle infinito durante su ejecución, salvo que se incluya algún mecanismo para evitarlo.

5. *Condiciones IF innecesarias.* Una condición IF innecesaria existe cuando dos reglas tienen idénticas conclusiones, una premisa de una regla está en contradicción con una premisa en la otra regla, y el resto de premisas son equivalentes.

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) \\ p(x) \wedge \neg q(x) &\rightarrow r(x) \end{aligned}$$

En este caso las dos reglas pueden resumirse en una que contemple sólo las premisas equivalentes.

$$p(x) \rightarrow r(x)$$

Puede haber situaciones especiales en las que una condición IF innecesaria no implique necesariamente que se unan las dos reglas. Así, por ejemplo, en el conjunto de reglas:

$$\begin{aligned} p(x) \wedge q(x) &\rightarrow r(x) \\ \neg q(x) &\rightarrow r(x) \end{aligned}$$

el problema quedaría resuelto eliminando la condición IF innecesaria de la primera regla, pero manteniendo intacta la segunda regla:

$$\begin{aligned} p(x) &\rightarrow r(x) \\ \neg q(x) &\rightarrow r(x) \end{aligned}$$

21.2.3.2 Verificación de la completitud

Frecuentemente ocurre que el proceso de adquisición de conocimiento a partir de fuentes de experiencia no es completo, lo que puede producir “huecos” en el conocimiento adquirido. Existen una serie de situaciones típicas que pueden ser indicativas de un “hueco” en la base de conocimientos, como son valores no referenciados de atributos, valores ilegales de atributos, reglas inalcanzables, o reglas “sin salida”.

1. *Valores no referenciados de atributos.* Esta situación ocurre cuando algunos valores, del conjunto de posibles valores de un atributo, no son cubiertos por la parte IF de ninguna regla. Por ejemplo, si tenemos el atributo Temperatura cuyo rango de posibles valores es {"alta", "normal", "baja"} de forma que los valores "alta" y "normal" aparecen en la parte IF de alguna regla, pero el valor "baja" no está representado en el antecedente de ninguna regla de la base de conocimientos, estamos ante una situación de valores no referenciados de atributos.

Un atributo parcialmente cubierto puede impedir que el sistema alcance una conclusión, o puede provocar conclusiones erróneas cuando el valor no cubierto aparece en la ejecución. Este error puede indicar la falta de alguna regla en la base de conocimientos.

2. *Valores ilegales de atributos.* Esta situación ocurre cuando una regla hace referencia a valores de atributos que no están incluidos en el conjunto de valores válidos para ese atributo. Por ejemplo, si el conjunto de valores válidos de Temperatura es {"alta", "normal", "baja"} y encontramos condiciones del tipo Temperatura = "muy alta" o Temperatura = "algo baja", bien en las premisas o en las conclusiones, estamos ante una situación de valores ilegales de atributos.

Este error es causado, generalmente, por equivocaciones en la escritura, aunque también puede ser indicativo de que el conjunto de valores válidos del atributo sea incompleto.

3. *Reglas inalcanzables.* En un sistema inteligente que utiliza una búsquedas regresiva (dirigida por las metas), una regla es inalcanzable si la conclusión de la regla no aparece en el objetivo a buscar, y no aparece en la condición IF de otra regla. Por ejemplo, la regla "IF Temperatura >37 THEN Fiebre" sería inalcanzable si "Fiebre" no apareciera en el objetivo buscado ni en la condición IF de otra regla.

En caso de que el razonamiento fuese dirigido por los datos, la regla sería inalcanzable si sus premisas no pudieran obtenerse del exterior (por ejemplo, preguntándole al usuario), y no aparecieran como conclusión de ninguna regla. Siguiendo con el ejemplo anterior, la regla “IF Temperatura >37 THEN Fiebre” sería inalcanzable si el valor de “Temperatura” no pudiera ser actualizado desde el exterior, ni apareciera como conclusión de alguna otra regla.

Esta situación puede afectar a la eficiencia del sistema pero nunca a su salida, ya que la regla con la conclusión inalcanzable nunca será disparada.

4. *Reglas sin salida.* Este caso está directamente relacionado con el analizado en el punto anterior. Una regla inalcanzable para un razonamiento dirigido por las metas es una regla “sin salida” para un razonamiento dirigido por los datos. De la misma forma, una regla inalcanzable para un razonamiento dirigido por los datos es una regla “sin salida” para un razonamiento dirigido por las metas.

21.2.4 Influencia de las medidas de incertidumbre

En sistemas que pretenden trabajar con incertidumbre también es importante verificar cuestiones como la consistencia, la completitud, la corrección, y la no redundancia. Estas tareas se realizan, en primer lugar, asegurándonos de que cada regla incluye un factor de certidumbre (o cualquier otra medida de características similares), y que estos factores cumplen las restricciones impuestas por la teoría o modelo en que se basan.

El modo en que el uso de medidas de incertidumbre puede afectar a la realización de los tests de consistencia y completitud puede verse en los siguientes ejemplos [Nguyen y otros, 1987]:

- *Redundancia.* Si antes la redundancia no afectaba a la salida del sistema ahora puede causar graves problema ya que, al considerar la misma información dos veces, pero con distintos valores, se pueden modificar los pesos de las conclusiones.
- *Reglas englobadas en otras.* Esta situación puede no ser errónea ya que las dos reglas pueden indicar la misma conclusión pero con distintas confianzas. La regla englobada sería un refinamiento de la regla más general para el caso de que tengamos más información.
- *Condiciones IF innecesarias.* Igual que en el caso anterior, una condición IF innecesaria puede utilizarse para variar la confianza en la conclusión final.
- *Reglas circulares.* Pueden existir casos en los que la utilización de medidas de incertidumbre rompan la circularidad de un conjunto de reglas. Por ejemplo, si el factor de certidumbre de una conclusión implicada en el ciclo cae por debajo de un umbral, por ejemplo, 0.2, se podría considerar que el valor de la conclusión es “desconocido” y el ciclo se rompe.

- *Reglas “sin salida”.* La detección de este tipo de reglas se complica con la introducción de incertidumbre. Así, una regla puede convertirse en una regla “sin salida” si su conclusión tiene una certidumbre por debajo del umbral en el cual un valor se considera “conocido”. Por ejemplo, la siguiente cadena de reglas:

$$A \xrightarrow[0,4]{R1} B \xrightarrow[0,7]{R2} C \xrightarrow[0,7]{R3} D$$

podría parecer válida, sin embargo si A se conoce con total certidumbre, el factor de certeza de D después de un razonamiento progresivo sería $0,4 \times 0,7 \times 0,7 = 0,196$, valor que cae por debajo del umbral antes mencionado. En este caso el valor de D sería “desconocido” y la línea de razonamiento acabaría en un punto “sin salida”.

- *Reglas inalcanzables.* De forma similar al ejemplo anterior pueden existir reglas que, por causa de los factores de certeza, se convierten en inalcanzables. Así, en el siguiente ejemplo:

$$A \xrightarrow[0,1]{R1} B \xrightarrow[1]{R2} C$$

la regla R_2 sería inalcanzable en un razonamiento progresivo (aunque su premisa aparece en la conclusión de otra regla) porque el valor de B cae por debajo del umbral de 0.2 (arbitrario).

21.3 Validación de sistemas inteligentes

Ya hemos mencionado que la validación consiste en comprobar si estamos construyendo el producto correcto; lo que requiere examinar la validez de los resultados suministrados por el sistema son correctos, y la constatación del cumplimiento de las necesidades y requisitos del usuario.

Antes de describir y proponer una metodología concreta, parece conveniente señalar y discutir algunas de las características principales del proceso de validación.

21.3.1 Personal involucrado

Una cuestión importante a determinar en todo tipo de validación es quién va a llevarla a cabo. El primer elemento a considerar es el ingeniero de conocimiento que ha desarrollado el sistema, ya que es quien mejor conoce las características del sistema inteligente. Sin embargo, la inclusión sin matices del ingeniero del conocimiento en el proceso puede afectar a la objetividad del estudio. Ello puede ser debido a que el ingeniero de conocimiento ha dedicado mucho esfuerzo al desarrollo del sistema y puede

sentirse inclinado a sobrevalorar los resultados del mismo. De todas formas, en la validación siempre es necesaria la presencia de una persona que tenga un conocimiento amplio del sistema, aunque no sea su constructor.

También es necesario contar con expertos humanos. Como veremos, el método básico para realizar la validación es el análisis de casos de prueba ya resueltos. Estos casos habrán sido analizados también por expertos humanos con los que podremos estudiar las discrepancias encontradas. Generalmente, es conveniente que los expertos que participen en la validación no sean los mismos que colaboren en el desarrollo del sistema. Con esta medida se intenta conseguir que el conocimiento del sistema se adecue al de un consenso de expertos, y no sólo al conocimiento del experto que ha colaborado en su diseño.

Relacionada con la necesidad de independencia en la validación surge la idea de hacer recaer todas las responsabilidades en un grupo de expertos independiente (que O'Keefe y O'Leary [O'Keefe y otros, 1987] denominan “terceros expertos”). Sin embargo, si el constructor del sistema podía sobrevalorar su producto, la participación de un grupo humano de validación totalmente independiente puede provocar el efecto contrario. Esta situación es la que Chandrasekaran [Chandrasekaran, 1983] describió como la “falacia del superhombre”: se le exige más al sistema inteligente de lo que se le exigiría a un experto humano, teniendo en cuenta que el conocimiento del sistema inteligente no es más que un modelo del conocimiento de los expertos humanos.

21.3.2 Qué validar

Nuestro principal objetivo es lograr que los resultados finales del sistema inteligente sean correctos. Sin embargo, también puede ser interesante analizar si los resultados intermedios del sistema son correctos, o si el razonamiento seguido hasta dar con la solución es apropiado.

La validación de los resultados intermedios puede ser interesante porque los resultados finales dependen de ellos. Así, el análisis de dichos resultados intermedios nos da una descripción del funcionamiento interno del sistema y nos permite una rápida corrección de los errores cometidos.

También puede resultar apropiado validar las estructuras de razonamiento; es decir, comprobar que el sistema alcanza la respuesta correcta por las razones correctas. Un proceso de razonamiento incorrecto puede provocar errores cuando queramos ampliar nuestra base de conocimientos. En este caso lo que se pretende es emular el proceso de razonamiento que realizan los expertos humanos. De esta forma, los usuarios del sistema encontrarán más agradable su utilización al seguir una línea lógica a la hora de plantear las cuestiones.

21.3.3 Casuística de validación

El uso de casos de prueba es el método más ampliamente utilizado para la validación de sistemas inteligentes. Al respecto, lo ideal sería poder contar con una gran cantidad de casos representativos de un espectro completo de problemas, casos que fuesen analizados por un grupo más o menos numeroso de expertos. En la realidad,

desafortunadamente, es muy común no disponer más que de un número reducido de casos y con pocos expertos que colaboren en su análisis. Para que una muestra de casos sea susceptible de ser aceptada en un proceso de validación debe cumplir dos propiedades fundamentales: cantidad y representatividad.

El número de casos empleados en la validación tiene que ser suficiente para que las medidas de rendimiento que obtengamos sean estadísticamente significativas. Ante esto podemos plantearnos un método muy sencillo de captura de datos: ir recogiendo todos los casos de que podamos disponer hasta que tengamos un número suficiente de ellos.

No obstante hay que considerar otra característica de la muestra como es su representatividad. No sólo hay que capturar un número elevado de casos, sino que éstos deben ser representativos de los problemas comunes a los que se va a enfrentar el sistema inteligente, aquellos problemas que constituyen su dominio. Chandrasekaran [Chandrasekaran, 1983] aconseja que aquellos problemas que resuelva el sistema inteligente deben aparecer representados en los casos de prueba. O'Keefe y otros [O'Keefe y otros, 1987] destacan que la cobertura de los casos es mucho más importante que su número. Así, los casos deben representar con fiabilidad el dominio de entrada del sistema. El dominio de entrada está constituido por aquellos casos que son susceptibles de ser tratados por el sistema inteligente. Cuanto mayor sea el dominio de entrada más compleja se hace la validación del sistema.

Para intentar mantener la representatividad de los datos se suelen emplear muestras estratificados. Supongamos que tenemos un Sistema Basado en Conocimiento (SBC) en medicina a partir del cual pretendemos discriminar entre tres posibles diagnósticos: A, B y C. Revisando la historia clínica comprobamos que en este tipo de clasificaciones el diagnóstico A ha aparecido el 50 % de las veces, B el 35 % y C el 15 %. De esta forma, si nuestra muestra está compuesta por 200 casos, 100 de ellos deben pertenecer al diagnóstico A, 70 al diagnóstico B y 30 al diagnóstico C.

El problema con las muestras estratificadas surge cuando una de las categorías sobrepasa en exceso a las otras. Siguiendo el ejemplo anterior ¿qué pasaría si el diagnóstico A apareciera en el 90 % de las veces? En este supuesto un sistema *trivial* que siempre diagnosticara esta categoría obtendría un 90 % de acierto. En dichas situaciones se recomienda utilizar muestras equilibradas, en la que existen los mismos casos de cada categoría. Posteriormente veremos cómo la medida kappa puede ser utilizada para corregir los índices de acuerdo en presencia de muestras poco balanceadas.

La casuística empleada en la validación del sistema debe incluir dos tipos de datos: por un lado las características de cada caso en particular y, por otro lado, un criterio que permita identificar el tipo de caso que estamos tratando. A continuación, el proceso de validación podría plantearse del siguiente modo (véase la Figura 21.2): (1) Obtención de la casuística de validación; (2) Los datos de la casuística son transferidos al sistema inteligente, que se encarga de interpretarlos, y (3) Los resultados del sistema y el criterio de validación que acompaña a los datos sirven de entrada para un proceso de validación en el que se analizará el rendimiento del sistema inteligente.

Por otra parte, la naturaleza del criterio de validación da lugar a distintos tipos de validación: validación contra el experto y validación contra el problema, cuyas características se detallan a continuación.

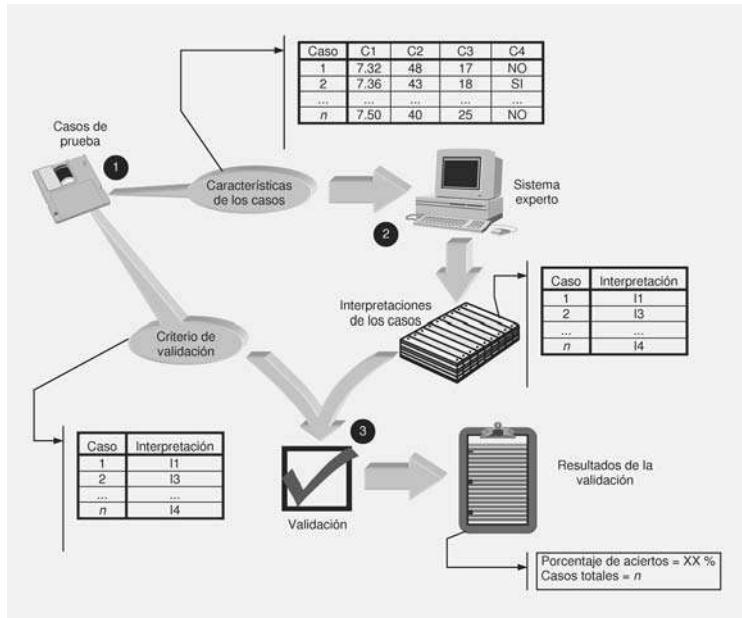


Figura 21.2: Proceso de validación a partir de casos de prueba: (1) Obtención de la casuística, (2) obtención de los resultados del sistema, y (3) proceso de validación.

21.3.4 Validación contra el experto

La validación contra el experto consiste, básicamente, en utilizar las opiniones y las interpretaciones de expertos humanos como criterio de validación. Este tipo de validación es el más comúnmente empleado en los sistemas inteligentes... después de todo, lo que pretendemos es construir un modelo del conocimiento del experto humano, por lo que resulta lógico utilizar a los expertos como una medida del desempeño de nuestro sistema en la validación. Existen tres posibles tipos de validación contra los expertos:

- 1. Validación contra un único experto.** La validación contra un único experto no es la más recomendable de todas pero, desgraciadamente, suele ser bastante común. Dada la escasa disponibilidad de expertos humanos, no siempre es posible contar con varios expertos en el proceso de validación. El inconveniente de utilizar un único experto es que la objetividad del estudio es cuestionable.
- 2. Validación contra un grupo de expertos.** Una situación más deseable a la hora de realizar la validación es contar con las opiniones de una serie de expertos humanos. Esto conlleva una serie de ventajas: (1) no estamos ligados a una única opinión, que puede ser errónea, y (2) permite comparar el grado de consistencia existente entre los expertos del dominio.

El principal inconveniente de esta técnica es cómo medir el rendimiento del sistema inteligente. Generalmente, los expertos no suelen tener la misma cualificación y se suele buscar una concordancia elevada con aquellos expertos de mayor nivel. Sin embargo, si los expertos son todos de un nivel similar normalmente lo que se busca es comprobar que las interpretaciones del sistema se parezcan a las de los expertos, tanto como las interpretaciones de los expertos se parecen entre sí. Para ello, existe una serie de medidas y procedimientos estadísticos potencialmente útiles para medir éstos, que analizaremos más adelante.

3. **Validación contra un consenso de expertos.** La otra opción comúnmente empleada en la validación con expertos, es conseguir unir las opiniones de varios expertos en una única opinión. Este consenso tiene la ventaja de que procura ser lo más objetivo posible y, si el acuerdo del sistema inteligente con el consenso es amplio, la confianza en el sistema aumentará considerablemente. El inconveniente de esta técnica es que, en cierta manera, estamos volviendo a la técnica de validación con un único experto, es decir, todo aquello que cae fuera del consenso es considerado erróneo. Sin embargo, puede haber otras soluciones válidas que los expertos podrían haber elegido, pero que han cambiado para adaptarse a un estándar del cual no están plenamente convencidos (posiblemente influidos porque un experto de mayor nivel está de acuerdo con el consenso). Además, la búsqueda de un estándar o consenso entre los expertos puede ser una ardua tarea. Entre los distintos métodos para lograr un consenso a partir de las opiniones de varios expertos destaca el método Delphi [Sackman, 1974].

21.3.5 Validación contra el problema

En la validación contra el problema se trata de descubrir si nuestro sistema resuelve realmente el problema planteado. La ventaja de este método de validación es evidente: se trata de un método completamente objetivo, la solución real del problema es la que se muestra. Si nuestro sistema discrepa del experto pero coincide con la solución real, la credibilidad del SBC se verá aumentada.

Sin embargo, este método también presenta inconvenientes. Uno de ellos es que podemos caer en la “falacia del superhombre”, es decir, exigirle más al SBC de lo que se le exigiría a un experto humano. Así, supongamos un sistema que presenta un acuerdo del 70 % con la solución real del problema. Este resultado puede parecer inadecuado, pero cuando analizamos los resultados de los distintos expertos humanos vemos que el acuerdo de estos con la solución real tampoco sobrepasa el 70 % y que sus interpretaciones son muy similares a las del SBC, podremos suponer que el comportamiento del SBC es aceptable, ya que su rendimiento es similar al de los expertos humanos del dominio.

Otro problema que surge en la validación contra el problema es que, puede no ser posible obtener una solución real. Por ejemplo, imaginemos un sistema médico que se encargue de aconsejar terapias para distintas patologías. Evidentemente, por razones éticas, sólo se podrá probar una terapia sobre un paciente si coincide con la terapia que ha prescrito el experto humano (lo que limita bastante el estudio).

21.4 Métodos cuantitativos de validación

La validación cuantitativa consiste en el empleo de medidas estadísticas para cuantificar el rendimiento de un SBC. Muchos métodos estadísticos se han empleado en la validación: contrastes de hipótesis, análisis de la varianza (ANOVA), intervalos de confianza, etc. Sin embargo, estas técnicas pueden resultar confusas para alguien que no tenga amplios conocimientos de estadística, y son difíciles de interpretar.

En nuestro estudio, hemos decidido centrarnos en aquellas técnicas más comunes y fáciles de interpretar que, utilizadas junto a otras medidas y técnicas gráficas, permiten tener un conocimiento amplio sobre el rendimiento de nuestro sistema.

Podemos dividir nuestras técnicas cuantitativas en tres grupos: medidas de pares, medidas de grupo y ratios de acuerdo. A continuación, veremos una descripción de las mismas y en el apartado de problemas resueltos veremos un ejemplo de su aplicación e interpretación.

21.4.1 Medidas de pares

Las medidas de pares pretenden evaluar el grado de acuerdo entre los resultados de dos expertos (incluyendo sistema inteligente, expertos humanos o una referencia estándar).

El desarrollo de una medida de pares parte de una base de datos de validación, en la que se incluyen los resultados del análisis de una serie de casos por parte de unos expertos. Cada experto realiza una evaluación de cada caso y asigna una etiqueta semántica determinada. El conjunto de etiquetas semánticas debe ser exhaustivo (tiene que existir una para cada posible caso), y las correspondientes etiquetas ser mutuamente excluyentes (a un caso sólo podemos asignarle una única etiqueta semántica).

Una vez tenemos nuestra base de datos de validación, las medidas de pares se deben obtener según el siguiente procedimiento: (1) se desarrolla una tabla o matriz de contingencia que relaciona los resultados de los expertos en consideración, y (2) se extrae de la tabla de contingencia la medida de pares determinada (véase la Figura 21.3).

21.4.1.1 Tablas de contingencia

Una tabla de contingencia es una tabla que relaciona de forma cruzada datos categóricos. En nuestro caso utilizaremos las tablas de contingencia para relacionar los resultados de dos expertos, como se puede ver en la Figura 21.4.

Esta tabla representa a dos expertos (A y B) que se encargan de asignar a cada caso una categoría semántica escogida de un conjunto de k posibles categorías. Cada celda de la tabla incluye una cantidad n_{ij} que representa el número de casos en los que el experto A selecciona la categoría i , mientras que el experto B selecciona la categoría j . Estas cantidades también se conocen con el término de frecuencias absolutas, o frecuencias absolutas observadas.

El número total de casos pertenecientes a la fila i se denota con el término n_i , mientras que el número total de casos de la columna j se denota con el término n_j .

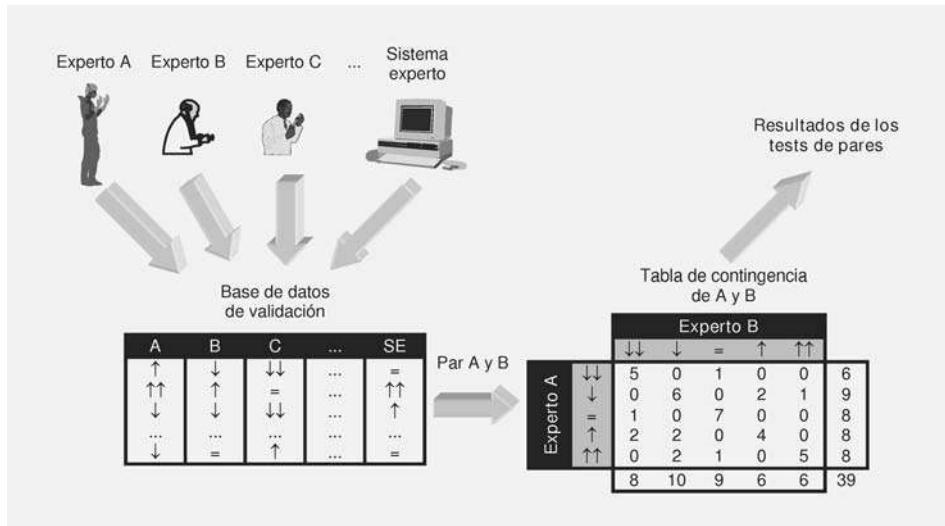


Figura 21.3: Esquema cualitativo de la IC. Se modela a partir de descripciones en lenguaje natural del procedimiento usado por un humano para resolver una tarea.

		Resultados experto B				Totales
		1	2	...	k	
Resultados experto A	1	n_{11}	n_{12}	...	n_{1k}	$n_{1\cdot}$
	2	n_{21}	n_{22}	...	n_{2k}	$n_{2\cdot}$

	k	n_{k1}	n_{k2}	...	n_{kk}	$n_{k\cdot}$
Totales		$n_{\cdot 1}$	$n_{\cdot 2}$...	$n_{\cdot k}$	$n_{\cdot \cdot} = N$

Figura 21.4: Tabla de Contigencia.

Estos valores se denominan frecuencias absolutas marginales (porque se suelen situar en los márgenes de la tabla), y se pueden obtener de los valores de las celdas de la siguiente forma:

$$\begin{aligned}
 n_{i\cdot} &= n_{i1} + n_{i2} + \dots + n_{ik} = \sum_{j=1}^k n_{ij} \\
 n_{\cdot j} &= n_{1j} + n_{2j} + \dots + n_{kj} = \sum_{i=1}^k n_{ij} \\
 n_{\cdot \cdot} &= \sum_{i=1}^k \sum_{j=1}^k n_{ij} = \sum_{i=1}^k n_{i\cdot} = \sum_{j=1}^k n_{\cdot j}
 \end{aligned}$$

donde el término $n_{\cdot \cdot}$ representa el número total de casos de la muestra y normalmente es indicado con la letra N . La notación empleada aquí se conoce como *notación de*

puntos, en la que un punto en un subíndice indica que se realiza una suma sobre ese subíndice. Las medidas de pares más utilizadas son las medidas de acuerdo, de las que aquí estudiaremos a continuación el índice de acuerdo, el índice de acuerdo dentro de uno, kappa y kappa ponderada.

21.4.1.2 Índice de acuerdo

Una de las medidas de acuerdo más comúnmente utilizada es el índice de acuerdo, o proporción de acuerdo. Se trata simplemente del cociente entre el número de observaciones de acuerdo y el número de observaciones totales. Para obtener esta medida a partir de la tabla de contingencia hay que sumar las frecuencias absolutas de la diagonal principal y dividirlas por el número total de casos, o sumar las frecuencias relativas o proporciones de la diagonal principal (representadas como p_{ij}).

$$\text{Índice de acuerdo} = \frac{\sum_{i=1, j=1, i=j}^k n_{ij}}{N} = \sum_{i=1, j=1, i=j}^k p_{ij}$$

El índice de acuerdo tiene las siguientes características: (a) toma valores en el intervalo $[0, 1]$ (en el que la unidad representa el acuerdo completo y el cero el desacuerdo completo), (b) su valor no se ve afectado por variaciones en el orden de las categorías, (c) un experto siempre presenta acuerdo perfecto consigo mismo, (d) el acuerdo perfecto es una relación transitiva, y (e) para su cálculo no importa el orden en el que escogamos a los expertos.

La principal ventaja de esta medida es la sencillez de su interpretación, que ha hecho que su uso se extendiera en distintos campos y aplicaciones. Sin embargo, presenta el inconveniente de que no tiene en cuenta los acuerdos debidos a la casualidad. Esta situación de concordancias o acuerdos por casualidad es muy frecuente cuando, por ejemplo, realizamos dos clasificaciones con distinto número de categorías (por ejemplo, una con categorías “muy bajo”, “bajo”, “normal”, “alto” y “muy alto”; y otra con categorías “muy bajo”, “normal”, “muy alto”). Generalmente, la clasificación con menor número de categorías tendrá un índice de acuerdo mayor, ya que la probabilidad de realizar asignaciones iguales por casualidad es mayor cuantas menos categorías haya. Otro inconveniente sería que se centra en los acuerdos, sin hacer diferencias entre los distintos desacuerdos que puedan aparecer.

21.4.1.3 Índice de acuerdo dentro de uno

Muchas veces, cuando utilizamos escalas ordinales en nuestras interpretaciones, es normal que se produzcan discrepancias que se diferencien sólo en una categoría lingüística. Por ello se utiliza el índice de acuerdo dentro de uno, que considera como acuerdos parciales aquellas interpretaciones que se diferencian en una única etiqueta lingüística consecutiva. En la Figura 21.5 podemos ver las interpretaciones de dos expertos y la representación gráfica de dichas interpretaciones.

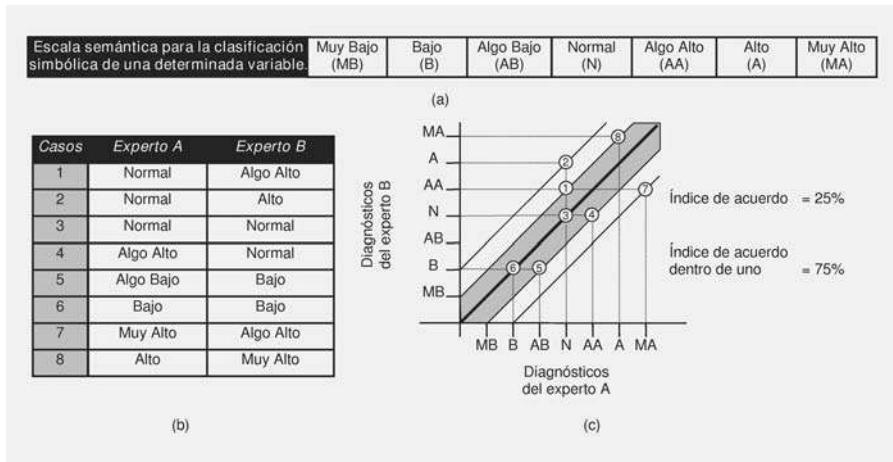


Figura 21.5: (a) Escala semántica para la clasificación simbólica de una determinada variable, (b) interpretaciones de dos expertos para esa variable y (c) representación de las desviaciones de esas interpretaciones siendo el área sombreada la representación del acuerdo dentro de uno.

Formalmente, podemos definir el índice de acuerdo dentro de uno de forma similar al índice de acuerdo pero, en este caso, sumando también las frecuencias pertenecientes a las diagonales adyacentes a la diagonal principal.

$$\text{Índice de acuerdo dentro de uno} = \frac{1}{N} \sum_{i=1, j=1, i=j, i=j \pm 1}^k n_{ij} = \sum_{i=1, j=1, i=j, i=j \pm 1}^k p_{ij}$$

El índice de acuerdo dentro de uno sólo es útil cuando estamos tratando escalas ordinales, pero tiene la ventaja de que puede utilizarse para analizar tendencias en los resultados. Sin embargo, presenta el mismo problema que el índice de acuerdo: no tiene en cuenta aquellos acuerdos que son debidos a la casualidad.

21.4.1.4 Índice kappa

Cohen [Cohen, 1960] propuso una medida de acuerdo (denominada kappa) en la que se corrían aquellos acuerdos que eran debidos a la casualidad. Esta medida está basada en dos cantidades:

- p_o = proporción de acuerdo observado.
- p_c = proporción de acuerdo esperado debido a la casualidad.

De esta forma, $1 - p_c$ representa el máximo acuerdo posible una vez que se ha eliminado la casualidad y $p_o - p_c$ representa el acuerdo observado una vez que se ha eliminado la casualidad. Esto nos permite definir el índice kappa de la siguiente forma:

$$\kappa = \frac{p_o - p_c}{1 - p_c}$$

El término p_o es el índice de acuerdo visto anteriormente, mientras que el término p_c es la suma de los productos de las proporciones marginales correspondientes a la diagonal principal y se expresa mediante la siguiente ecuación:

$$p_c = \sum_{i=1, j=1, i=j}^k p_i p_j$$

La ecuación anterior se obtiene siguiendo los tres pasos que se representan en la Figura 21.6 (que representa la tabla de contingencia que relaciona las interpretaciones de dos expertos):

1. Cálculo de las proporciones marginales mediante la suma de las proporciones correspondientes a las filas y a las columnas.
2. Cálculo de las proporciones de acuerdo debidas a la causalidad de cada celda mediante la multiplicación de las proporciones marginales correspondientes a dicha celda.
3. Cálculo del índice p_c mediante la suma de las proporciones de acuerdo debidas a la causalidad de las celdas pertenecientes a la diagonal principal.

		Experto 1				
		Muy Bajo	Bajo	Normal	Alto	Muy Alto
Experto 2	Muy Bajo	2 0 (0)	0,1	0	0	0
	Bajo	0	0,2 (0,12)	0,1	0	0,3
	Normal	0	0	0,3 (0,12)	0	0,3
	Alto	0	0,1	0	0,1 (0,02)	0,2
	Muy Alto	0	0	0	0	0,1 (0,01)
		0	0,4	0,4	0,1	0,1 3 1

Figura 21.6: Pasos para el cálculo del índice p_c .

En este caso en concreto el valor de p_o sería 0,7 y el valor de p_c sería de $0 + 0,12 + 0,12 + 0,02 + 0,01 = 0,27$. El valor final del índice kappa sería, por tanto: $(0,7 - 0,27)/(1 - 0,27) = 0,589$.

La idea que subyace bajo estos cálculos es la siguiente: las proporciones marginales representan la distribución de las interpretaciones de los dos expertos cuando son tratados de forma independiente. Cuando se comparan sus interpretaciones se espera que su acuerdo sea mayor que el que cabría esperar si tomamos sus proporciones marginales como probabilidades (así, la probabilidad de que escoja una determinada

celda es el producto de las proporciones marginales correspondientes a dicha celda). Por ello se corrige el acuerdo observado (p_o) con el acuerdo debido a la casualidad (p_c). Si el acuerdo observado es igual al acuerdo debido a la casualidad el valor de kappa es cero. Si el acuerdo observado es mayor que el acuerdo debido a la casualidad el valor de kappa es positivo, siendo su límite máximo +1 (que corresponde al acuerdo perfecto, $p_o = 1$). Si el acuerdo observado es menor que el debido a la casualidad el valor de kappa es negativo (nótese que el valor de kappa no está definido si $p_c = 1$). Sin embargo, su límite inferior no es tan sencillo de determinar como su límite superior, ya que depende de las distribuciones marginales.

El índice kappa es útil para corregir aquellas situaciones en las que los datos utilizados en la validación no se distribuyen uniformemente entre las distintas categorías. Así, en los datos de la Figura 21.6 vemos cómo un valor del índice de acuerdo de 0,7 es corregido por kappa hasta el valor 0,589. Esta disminución se debe a que los datos de origen se concentran en unas pocas categorías (bajo y normal) por lo que un sistema que interpretara dichas categorías con más frecuencia que las otras tendría más posibilidades de acertar con el diagnóstico correcto “por casualidad”.

21.4.1.5 Kappa ponderada

Kappa ponderada (κ_w) fue desarrollada también por Cohen [Cohen, 1968] y es una medida de acuerdo que corrige aquellos acuerdos debidos a la casualidad, y pondera de forma distinta los desacuerdos encontrados.

La ponderación de los distintos desacuerdos se hace a partir de una matriz de pesos en la que, para cada posible par de categorías ij , se define un peso v_{ij} , que cuantifica el desacuerdo existente. A las celdas pertenecientes a la diagonal principal (que representan el acuerdo perfecto) se les suele asignar el valor 0 indicando que no existe ningún desacuerdo. El mayor valor de desacuerdo v_{max} es fijado por el investigador. Para cualquier conjunto de pesos, kappa ponderada es invariable ante transformaciones multiplicativas positivas, es decir, que kappa ponderada no cambiará de valor si sus pesos se multiplican por un valor mayor que cero.

Para incluir los pesos en la ecuación de kappa procedemos de la siguiente manera: en primer lugar partimos de la ecuación en la que definímos el índice kappa a partir de la proporción de acuerdo observado y la proporción de acuerdo debido a la casualidad. Esta expresión también puede obtenerse en base a proporciones de desacuerdo ($q = 1 - p$) en donde la proporción de desacuerdo observado es $q_o = 1 - p_o$ y la proporción de desacuerdo debido a la casualidad es $q_c = 1 - p_c$. Sustituyendo p_o y p_c por $(1 - q_o)$ y $(1 - q_c)$ en la ecuación que define a kappa obtenemos:

$$\kappa = \frac{q_c - q_o}{q_c} = 1 - \frac{q_o}{q_c}$$

Esta ecuación expresa el coeficiente kappa basado en proporciones de desacuerdo. Para el cálculo de κ_w se reemplazan dichas proporciones de desacuerdo por las proporciones de desacuerdo ponderado q'_o y q'_c que se definen como:

$$q'_o = \frac{\sum_{i=1,j=1}^k v_{ij} p_{oj}}{v_{max}}, \quad q'_c = \frac{\sum_{i=1,j=1}^k v_{ij} p_{cj}}$$

en donde p_{oj} es la proporción de acuerdo observada para la casilla ij , p_{cj} es la proporción de acuerdo debido a la casualidad correspondiente a la casilla ij , v_{ij} es el peso correspondiente a la casilla ij , v_{max} es el peso máximo de la tabla y k es el número de categorías.

Usando ahora los valores de q'_o y q'_c y eliminando el valor de v_{max} , que aparece en el numerador y en el denominador, podemos definir kappa ponderada como:

$$\kappa_w = 1 - \frac{\sum_{i=1,j=1}^k v_{ij} p_{oj}}{\sum_{i=1,j=1}^k v_{ij} p_{cj}}$$

Podemos considerar a kappa como un caso especial de kappa ponderada en donde los $k(k - 1)$ pesos que no pertenecen a celdas de la diagonal principal tienen un valor constante mayor que el valor que tienen las celdas de la diagonal principal (que normalmente será cero para pesos de desacuerdo). Si se da esta situación es fácil ver cómo la ecuación que define a kappa ponderada se simplifica en la ecuación que define a kappa.

21.4.2 Medidas de grupo

Hasta ahora hemos visto cómo medir el acuerdo entre un par de expertos. Estos “expertos” pueden ser expertos humanos, un sistema inteligente, la opinión consensuada de un grupo de expertos o, incluso, la solución real al problema planteado. Sin embargo, los estándares son muy difíciles de encontrar cuando tratamos de validar un SBC. Lo normal será disponer de las opiniones de varios expertos humanos y de los resultados del sistema inteligente. Con estos datos trataremos de determinar, a través de diversas medidas de grupo, si los resultados del sistema inteligente son similares a los de los expertos humanos.

El proceso fundamental de aplicación de las medidas de grupo es el que se muestra en la Figura 21.7, en la que vemos cómo, en primer lugar, se obtiene la base de datos de validación con la que se comparan los resultados de los distintos expertos para una serie de casos seleccionados. Posteriormente, se llevan a cabo los tests de pares ya descritos, y sus resultados se detallan en una serie de tablas resumen (una por cada test de pares seleccionado), en la que se incluyen los resultados obtenidos por todos los pares posibles de expertos. Basándose en esta información podemos llevar a cabo distintos tests de grupo como las medidas de Williams o el análisis de agrupamientos.

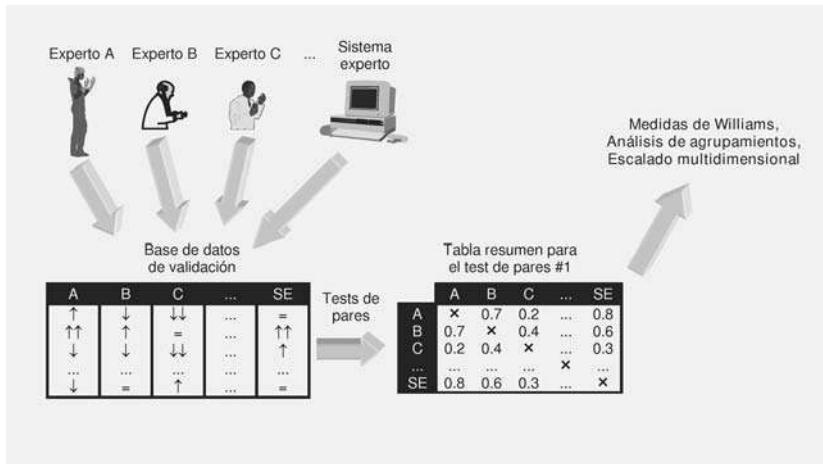


Figura 21.7: Proceso de realización de las medidas de grupo.

21.4.2.1 Medidas de Williams

En la literatura podemos encontrar diversos métodos para medir el acuerdo entre dos o más expertos (o evaluadores) cuyas respuestas son de tipo categórico (ordinales o nominales). [Fleiss, 1971] generalizó el estadístico kappa propuesto por [Cohen, 1960] para el caso de una muestra de N casos que es evaluada por n expertos. [Light, 1971] también consideró el problema de medir el acuerdo entre dos o más expertos y propuso otra medida que es una extensión de la medida kappa.

Sin embargo, en muchas situaciones podemos no estar interesados en comparar la consistencia interna de dos o más expertos. En vez de ello, podemos aislar un determinado experto y comparar las respuestas del experto aislado con las respuestas del resto del grupo. [Light, 1971] consideró el problema de comparar el acuerdo de varios expertos con un conjunto “correcto” de respuestas.

[Williams, 1976] trató un problema similar al de Light pero desde una óptica completamente distinta. Si asumimos que no disponemos de un conjunto “correcto” de respuestas, el objetivo de Williams es decidir si, dado un grupo de expertos y un experto aislado, el acuerdo entre el experto aislado y el grupo era similar al acuerdo existente entre dos miembros cualesquiera del grupo. Para ello parte, en primer lugar, de una medida de pares $P_{(a,b)}$ que nos da la similitud entre dos expertos (Williams utiliza el índice de acuerdo pero es posible utilizar cualquier otra medida de pares).

Basándonos en este coeficiente podemos definir el acuerdo existente dentro del grupo de expertos como la media de los acuerdos existentes entre los posibles pares de expertos que pertenecen a dicho grupo (denotado como P_n):

$$P_n = 2 \frac{\sum_{a=1}^n \sum_{b=a+1}^n P_{(a,b)}}{n(n-1)}$$

El acuerdo total del experto aislado con el grupo de referencia puede ser medido por la media de los distintos $P_{(0,a)}$, siendo $a = 1, \dots, n$, de la siguiente forma:

$$P_0 = \frac{\sum_{a=1}^n P_{(0,a)}}{n}$$

De esta forma podemos definir el índice de Williams como:

$$I_0 = \frac{P_0}{P_n}$$

La interpretación de este índice es la siguiente:

- Si I_0 es menor que uno indica que el acuerdo entre el experto aislado y el grupo de expertos es menor que el acuerdo entre los propios miembros del grupo.
- Si I_0 es igual a uno indica que el experto aislado coincide con el grupo al mismo nivel que los miembros del grupo coinciden entre sí.
- Si I_0 es mayor que uno indica que el experto aislado está más de acuerdo con el conjunto de expertos que como lo están los miembros del grupo entre sí. En este caso podría decirse que el experto aislado coincide con el consenso del grupo de expertos.

21.4.2.2 Análisis de agrupamientos

En las medidas de Williams analizadas en el apartado anterior veíamos cómo se intentaba determinar si los resultados de un experto aislado eran similares a los resultados de un grupo de expertos tomados como referencia. Un tratamiento parecido sería clasificar a los expertos en grupos según la similitud de sus interpretaciones, y comprobar si nuestro experto aislado pertenece (o está cerca) de aquel grupo formado por los expertos de mayor experiencia.

El análisis de agrupamientos se puede definir como el estudio formal de métodos y algoritmos para organizar objetivamente datos numéricos [Dubes, 1993] (véase el capítulo 16). En el análisis de agrupamientos el investigador no tiene conocimiento a priori sobre la estructura subyacente de los datos (o, si lo tiene, es un conocimiento limitado). Esto permite diferenciar claramente el análisis de agrupamientos del análisis discriminante. En este último el investigador sí conoce a priori el esquema de clasificación, y lo que intenta es dilucidar las reglas que determinan la pertenencia de un elemento a un grupo basándose en una serie de variables. Para ello, suele disponer de datos de entrenamiento para cada uno de los grupos especificados. El análisis de agrupamientos también puede verse como una técnica de reducción de datos. Los elementos disponibles se agrupan en distintos grupos de tal forma que los perfiles de los individuos que están en un mismo grupo son similares.

En el caso de la validación los datos de inicio con los que contamos para realizar un análisis de agrupamientos son las tablas resumen que indican, para cada par de expertos, el valor de una medida de pares. Tomando esta medida de pares como una

medida de similitud podemos proceder a agrupar los expertos. Las técnicas de agrupamientos pueden agruparse en dos grandes grupos: técnicas jerárquicas y técnicas iterativas. Utilizando las técnicas de agrupamientos jerárquico podemos ir formando grupos en pasos sucesivos, construyendo el árbol de partición o dendrograma (véase la Figura 21.8), que permite visualizar los resultados.

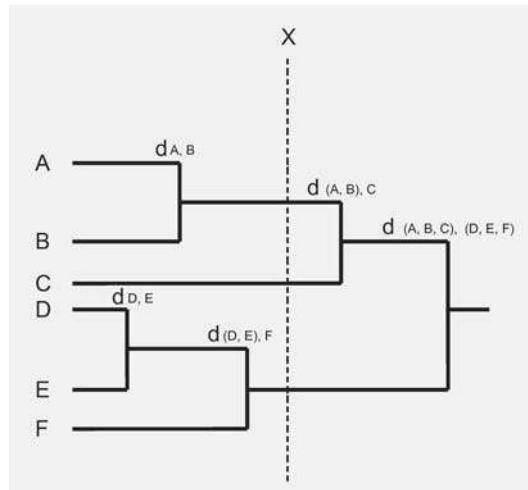


Figura 21.8: Esquema cualitativo de la IC. Se modela a partir de descripciones en lenguaje natural del procedimiento usado por un humano para resolver una tarea.

Así, en la Figura 21.8, podemos ver un dendrograma correspondiente al agrupamiento de 6 individuos. Si particionamos el árbol en el nivel X obtenemos los grupos cuya distancia entre cualquier par de expertos dentro de un mismo grupo son menores que C (en este caso obtenemos A-B, C y D-E-F). Variando X obtenemos distintas particiones del conjunto dado.

21.4.3 Ratios de acuerdo

El último tipo de medidas que estudiaremos son los llamados ratios de acuerdo. Los ratios de acuerdo tratan de medir el acuerdo existente entre un experto (o sistema inteligente), y una referencia estándar. Dicha referencia puede ser un consenso existente entre los expertos, lo que implicaría una validación contra el experto, o la solución real al problema planteado, lo que implicaría una validación contra el problema.

El cálculo de los ratios de acuerdo se basa en la construcción de una tabla de contingencia 2×2 para cada una de las categorías en las que se divide una interpretación dada (véase la Figura 21.9). En esta tabla se relacionan los resultados del experto (o del SBC) con los resultados de la referencia estándar para esa categoría en particular.

		Referencia estándar		
		D	$\neg D$	
Sistema experto	D	a	b	$a + b$
	$\neg D$	c	d	$c + d$
		$a + c$	$b + d$	$a + b + c + d$

Figura 21.9: Tabla de Contingencia para calcular los ratios de acuerdo.

Los valores a , b , c y d tienen el siguiente significado:

- a : Verdaderos positivos
- b : Falsos positivos
- c : Falsos negativos
- d : Verdaderos negativos

Sin embargo, estos valores no suelen analizarse directamente. Más bien la validación se centra en el cálculo de una serie de ratios derivados de estos valores:

1. **Índice de acuerdo.** El índice de acuerdo representa la proporción de casos en los que el SBC ha coincidido con el estándar para la categoría tomada en consideración. Es importante no confundirlo con el índice de acuerdo de los tests de pares, en los que se consideraban todas las posibles categorías.

$$\text{Índice de Acuerdo} = \frac{a + d}{a + b + c + d}$$

2. **Sensibilidad.** La sensibilidad se define como el ratio de verdaderos positivos y, tal y como se muestra en la ecuación, se calcula como el número de veces que hemos interpretado correctamente la categoría considerada, dividido por el número total de veces que aparece dicha categoría en el estándar.

$$\text{Sensibilidad} = \frac{a}{a + c}$$

La sensibilidad nos permite medir la capacidad del SBC para clasificar correctamente los casos positivos, y puede entenderse como la probabilidad de que el SBC clasifique correctamente el caso una vez sabido que es positivo. Un valor relacionado con la sensibilidad es el ratio de falsos negativos, que se calcula como el número de veces que erróneamente no hemos interpretado la categoría considerada partido por el número de veces que dicha categoría aparecía en el estándar.

$$\text{Ratio Falsos Negativos} = 1 - \text{Sensibilidad} = \frac{c}{a + c}$$

3. **Especificidad.** La especificidad se define como el ratio de verdaderos negativos y, tal y como se muestra en la ecuación, se calcula como el número de veces que hemos interpretado correctamente la ausencia de la categoría seleccionada, dividido por el número de veces que dicha categoría no aparecía en el estándar.

$$\text{Especificidad} = \frac{d}{b + d}$$

La especificidad nos permite medir la capacidad del sistema inteligente para clasificar correctamente los casos negativos y puede entenderse como la probabilidad de que el sistema inteligente clasifique correctamente el caso sabiendo que es negativo. Un valor relacionado con la especificidad es el ratio de falsos positivos, que se calcula como el número de veces que erróneamente hemos interpretado la categoría considerada, dividido por el número de veces que dicha categoría no aparecía en el estándar.

$$\text{Ratio Falsos Positivos} = 1 - \text{Especificidad} = \frac{b}{b + d}$$

4. **Valores predictivos.** Además de la sensibilidad y la especificidad también son muy utilizados los valores predictivos, tanto positivos como negativos. El valor predictivo positivo se define como el número de veces que hemos interpretado correctamente la categoría considerada, dividido por el número total de veces que dicha categoría aparecía en nuestros resultados.

$$\text{Valor Predictivo Positivo} = \frac{a}{a + b}$$

De la misma forma el valor predictivo negativo se define como el número de veces que hemos interpretado la ausencia de la categoría considerada, dividido por el número total de veces que dicha categoría no aparecía en nuestros resultados.

$$\text{Valor Predictivo Negativo} = \frac{d}{c + d}$$

Los valores predictivos pueden interpretarse como la probabilidad de que un caso sea positivo/negativo sabiendo que el test lo ha interpretado como positivo/negativo.

5. **Medida de Jaccard.** Un coeficiente muy usado en las tablas de contingencia 2×2 es la medida de Jaccard [Jaccard, 1908] que se define según la ecuación:

$$\text{Jaccard} = \frac{a}{a + b + c}$$

La medida de Jaccard elimina, tanto del numerador como del denominador, el número de casos en los que el experto y el estándar no han diagnosticado la

categoría considerada (es decir, eliminan el número d). La medida de Jaccard es muy útil para medir acuerdos en aquellas situaciones en las que resultan más importantes los resultados positivos que los negativos y en situaciones en las cuales el número de casos negativos es mayor que el número de casos positivos. Por ejemplo, si tratamos de establecer la similitud entre dos especies de mamíferos nos centraremos en comprobar las características que comparten (dieta, hábitat, etc.) y de nada nos servirá saber que ambas especies coinciden en no tener agallas, alas, piel escamosa, etc.

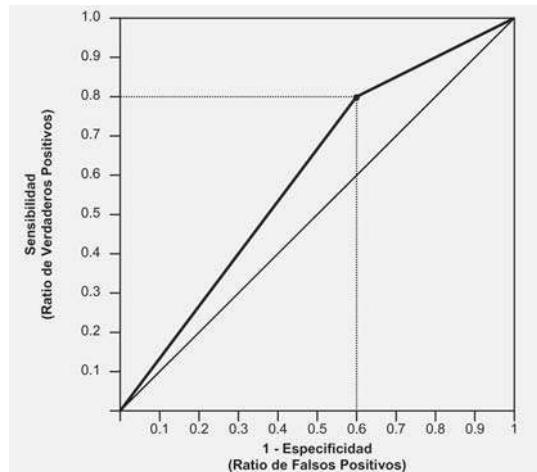


Figura 21.10: Ejemplo de curva ROC de un solo punto.

21.4.3.1 Curvas ROC

Las curvas ROC (Receiver Operating Characteristic) son representaciones gráficas que relacionan el ratio de verdaderos positivos (sensibilidad) con el ratio de falsos positivos (1-especificidad) tal y como se muestra en la Figura 21.10. La curva ROC parte del origen de coordenadas hasta la esquina superior derecha pasando por el punto en el que se cruzan los valores de la sensibilidad y 1-especificidad.

Las curvas ROC se utilizan para analizar el rendimiento tanto en casos positivos como en casos negativos. Para ello se calcula el área situada bajo la curva y que denominamos θ . Dicha área representa la probabilidad de una respuesta acertada, mientras que la cantidad $(1 - \theta)$ representa la tasa de respuestas incorrectas producidas. Para que el área sea lo más grande posible el “codo” de la curva ROC debe situarse lo más cerca posible de la esquina superior izquierda. En una curva ROC de un solo punto es fácil comprobar que:

$$\text{Área bajo ROC} = \theta = \frac{\text{Sensibilidad} + \text{Especificidad}}{2}$$

Las curvas ROC también se utilizan para analizar cómo un determinado criterio de decisión interno afecta al rendimiento del sistema. Para descubrir esta influencia se computan los ratios de acuerdo para varias situaciones en las que se ha variado el criterio de decisión interno sobre su posible rango y se forma la curva ROC uniendo estos puntos (véase la Figura 21.11). Después sólo hay que elegir el punto de la curva que mejor se adecue a los objetivos planteados. Por ejemplo, podemos tolerar cierto grado de falsos positivos porque lo que verdaderamente nos interesa es maximizar los verdaderos positivos.

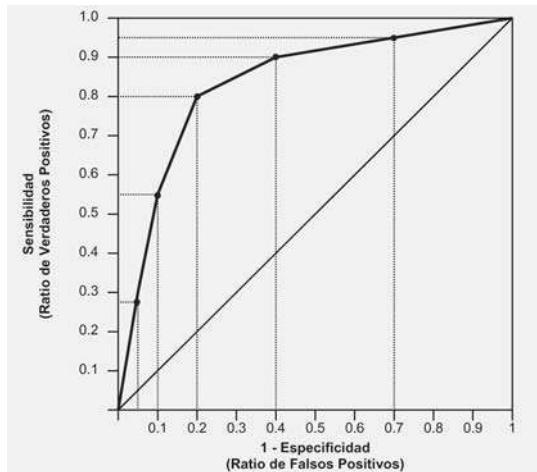


Figura 21.11: Ejemplo de curva ROC de varios puntos.



Figura 21.12: Separación entre valores negativos y positivos de una muestra. La elección ideal del punto de corte será aquella que minimice los falsos positivos y negativos.

En la mayoría de las ocasiones intentar incrementar el ratio de verdaderos positivos se hará a costa de incrementar el ratio de falsos positivos, y decrementar el ratio de falsos positivos se hará a costa de decrementar el ratio de verdaderos positivos. Esto es debido a que, tal y como se ve en la Figura 21.12, los casos positivos no están claramente diferenciados de los casos negativos. Lo que se intenta es buscar el valor

del criterio de decisión que represente el mejor compromiso entre el ratio de verdaderos positivos y el ratio de falsos positivos, es decir, que esté lo más cerca posible de la esquina superior izquierda en la curva ROC.

21.5 Síntesis metodológica del proceso de validación

Uno de los principales problemas que surge a la hora de realizar la validación de un sistema inteligente es la falta de una metodología estándar. Generalmente, todas las validaciones se suelen realizar de una manera informal, lo que dificulta la interpretación y comparación de los resultados.

En este apartado proponemos una metodología [Mosqueira-Rey y Moret-Bonillo, 2000] que se basa en la caracterización del proceso de validación, y en la utilización de medidas cuantitativas. La metodología se compone de tres partes claramente diferenciadas: planificación, aplicación, e interpretación, que desarrollaremos a continuación.

21.5.1 Planificación del proceso

Esta fase es necesaria porque, el dominio de aplicación del sistema, las propias características del sistema, y la fase de desarrollo en la que se encuentra, pueden motivar la utilización de determinados paradigmas de validación en detrimento de otros. La fase de planificación es una fase netamente heurística en la que la experiencia previa del ingeniero de conocimiento es determinante para el correcto establecimiento de un plan de validación. Para su diseño hay que tener en cuenta los siguientes aspectos:

- **Influencia del dominio de aplicación.** Las características del dominio de aplicación en el que se va a utilizar el sistema tienen un importancia fundamental a la hora de determinar los procesos de validación. El principal inconveniente con que se enfrenta la mayoría de los sistemas inteligentes y que, en cierto sentido, los aleja del software convencional, es que en la mayoría de las ocasiones se utilizan en entornos denominados críticos. En dichos dominios el coste de una decisión errónea es muy elevado por lo que el proceso de validación debe ser más riguroso. Además, un dominio crítico puede limitar las técnicas a emplear en la validación. De esta forma, la realización de pruebas prospectivas y tests de campo están muy limitadas cuando nos movemos en dichos entornos, y sólo podremos realizarlas en sistemas que no exijan una manipulación del entorno (por ejemplo, sistemas de predicción).
- **Criterio de validación.** Existen dos posibles criterios para validar un SBC; realizar una validación contra los expertos, o realizar una validación contra el problema. Sin embargo, la mayoría de las veces el propio dominio de aplicación limita la posibilidad de elección del criterio de validación. En dominios en los que la disponibilidad de los expertos es escasa, es difícil conseguir que un grupo de ellos colaboren en la validación de nuestro sistema. En tales casos lo más probable será contar con un único experto, por lo que la objetividad de nuestro estudio puede quedar en entredicho. Si la disponibilidad de expertos no es un

problema, lo más adecuado es realizar la validación contra un grupo de expertos, o utilizar dicho grupo de expertos para realizar un consenso que pueda ser utilizado como estándar en la validación. En cuanto a la validación contra el problema, no siempre es posible obtener un estándar que nos indique que el problema se ha resuelto correctamente. La validación orientada al problema se utiliza sobre todo en tareas de pronóstico, en donde es posible conocer si el resultado del pronóstico fue correcto o no.

- **Perfil del usuario final.** En el proceso de validación pueden estar involucrados: el ingeniero del conocimiento, expertos del dominio, evaluadores independientes y usuarios finales. Los usuarios finales no suelen tener una participación activa en el proceso de validación orientado a los resultados. Sin embargo, si los usuarios son expertos del domino puede ser común utilizarlos para realizar técnicas como los tests de campo. Si los usuarios no son expertos, normalmente sólo podrán colaborar en una validación orientada al uso. [Lamberti y Newsome, 1989] descubrieron que el rendimiento de usuarios no expertos aumentaba cuando eran enfrentados a cuestiones concretas, en vez de a representaciones abstractas.
- **Influencia del sistema.** Las características del sistema que va a ser validado también influyen en la forma a llevar a cabo dicha validación. La validación de subsistemas sólo podrá llevarse a cabo si es posible dividir nuestro sistema inteligente en módulos independientes. Si los módulos actúan como salidas y entradas unos de otros, o si su interacción es elevada, puede no ser posible validarlos por separado.
- **Manejo de incertidumbre.** Si el sistema maneja medidas de certidumbre es necesario contemplarlas en el proceso de validación. Una forma típica de realizar la validación en presencia de medidas de certidumbre es efectuando estudios de sensibilidad, en los que se realizan pequeños cambios en los datos de entrada y se estudia su efecto en las salidas del SBC.
- **Tipo de problema tratado.** El tipo de problema tratado también influye en la metodología de validación. Podemos distinguir dos tipos principales de problemas: (1) problemas de análisis, en los que a partir de una serie de casos, las interpretaciones son asignadas dentro de una determinada categoría (pueden ser problemas de diagnóstico, problemas de predicción, etc.), y (2) problemas de síntesis, en donde el resultado incluye la realización de un determinado plan de acción (por ejemplo, la elaboración de un determinado plan terapéutico).

Los sistemas que tratan problemas de análisis son más fáciles de validar ya que es posible aplicarlos sobre casos históricos y es más fácil de encontrar referencias estándar para dicha validación.

Los sistemas que tratan problemas de síntesis encuentran más dificultades en la validación. Así, validar un determinado plan generalmente suele implicar la actuación directa sobre el dominio de aplicación para ver su evolución, ya que no es fácil que existan estándares para probar su validez. Todo esto se complica si el dominio es crítico.

- **Relación con el entorno.** Generalmente los sistemas inteligentes no actúan de forma aislada, sino que se encuentran integrados en un sistema mayor (por ejemplo, el sistema de información de un hospital). En tal caso la validación de los interfaces con los otros elementos que forman el entorno (bases de datos, sistemas de entrada y salida, etc.) es fundamental para evaluar el correcto funcionamiento del sistema. Si el sistema no se integra en uno mayor, y actúa de forma autónoma, su validación se hace más sencilla.
- **Influencia de la fase de desarrollo.** La fase en la que se encuentre el desarrollo del sistema también influirá en la forma de realizar la validación. Cuanto más avanzado esté el desarrollo del sistema más compleja será la verificación y validación del mismo. Así, en las primeras etapas la validación consistirá en simples pruebas con casos ya resueltos, realizadas en un ambiente de laboratorio y que sólo cubrirán aspectos concretos del sistema. Posteriormente, el número de casos analizados por el sistema será mayor y también lo será su cobertura, la validación no será llevada a cabo sólo por el ingeniero del conocimiento, sino que entrarán a formar parte de la misma expertos ajenos al desarrollo del sistema. Por último, en las fases finales primarán más los aspectos orientados al uso y se preparará el terreno para una validación en el entorno de trabajo por parte de los usuarios finales.

21.5.2 Fase de aplicación de técnicas

La salida de la fase de planificación está constituida por una serie de estrategias que nos indican cómo realizar la validación del sistema inteligente que estamos considerando. La fase de aplicación será la encargada de llevar a la práctica estas recomendaciones obteniendo unos resultados que luego puedan ser interpretados en la siguiente fase. La fase de aplicación se compone de los siguientes pasos:

1. **Captura de la casuística.** Para realizar la validación es necesario contar con una serie de casos ya resueltos con los que podamos comparar los resultados de nuestro sistema inteligente. Como ya se ha comentado, es necesario que la muestra obtenida cumpla dos características fundamentales: cantidad y representatividad. La muestra deberá ser suficientemente amplia y debe cubrir todos los aspectos que pretendemos probar en nuestro sistema. La utilización de casos de prueba en la validación se adapta perfectamente a los métodos de desarrollo incremental, aumentando el número de casos y su cobertura a medida que avanzamos en el desarrollo.
2. **Preprocesado de los datos.** Una vez capturada la casuística tenemos una base de datos en la que se comparan los resultados de una o varias referencias (expertos humanos o la solución real del problema) con los resultados de nuestro SBC. Sin embargo, la base de datos obtenida generalmente no se puede utilizar directamente en los procesos de validación, y es necesario llevar a cabo un preprocesado que puede incluir: corrección de errores, transformación de los

datos o inclusión de información adicional. Famili [Famili y otros, 1996] destaca la importancia de la correcta selección de las técnicas de preprocesado para encontrar información útil durante el análisis.

3. **Realización de medidas estadísticas.** Las estrategias de validación obtenidas en el proceso de planificación indican qué medidas estadísticas son las más adecuadas a utilizar en función de las características del dominio y del sistema. Las medidas estadísticas incluidas en esta metodología son de tres tipos: medidas de pares, medidas de grupo y ratios de acuerdo.

21.5.3 Interpretación de resultados

La fase de interpretación se basa en los resultados de la fase de aplicación para dilucidar si el sistema inteligente se comporta realmente como un experto dentro de su campo de aplicación. La interpretación de los resultados se compone básicamente de dos tareas:

- **Análisis algorítmico.** El objetivo del análisis algorítmico es el de tratar los resultados de validación, teniendo en cuenta el contexto, para obtener información susceptible de ser tratada por el módulo heurístico de validación. Este módulo se denomina algorítmico porque está formado por funciones y algoritmos de tratamiento de datos que no incluyen conocimiento heurístico.
- **Análisis heurístico.** El análisis heurístico utiliza la información obtenida por el análisis algorítmico para realizar su interpretación a través de una serie de heurísticas que capturan el comportamiento inteligente del ingeniero del conocimiento a la hora de interpretar los resultados de validación.

Como ejemplo de realización de la fase de interpretación tengamos en cuenta el proceso que se muestra en la Figura 21.13. En dicha figura vemos que existen diez casos (evidentemente escasos para una validación representativa pero que nos sirven para ilustrar el ejemplo), evaluados por un experto humano (A) y por un SBC (SE). Esta base de datos de validación se pasa por la fase de aplicación para obtener una serie de resultados, en este caso se muestran los valores obtenidos por los tests de pares.

Ahora es necesario interpretar esta información, por ello en primer lugar la fase algorítmica convierte los resultados de la fase de aplicación en información teniendo en cuenta el contexto particular en el que estamos trabajando. En este contexto se incluirá información del dominio y las reglas que permitan la clasificación simbólica de los distintos valores obtenidos después de la fase de aplicación. Como resultado de este análisis algorítmico obtenemos que el valor del índice de acuerdo es alto, el valor de kappa es medio y el valor de kappa ponderada es bajo. La información obtenida del análisis algorítmico pasa a la parte heurística de la fase de interpretación. El análisis heurístico contendrá reglas del tipo:

IF: (IND_ACUERDO = ALTO)
AND: ((KAPPA_POND = BAJO) OR (KAPPA_POND = MEDIO))
THEN: MUESTRA_INCORRECTAMENTE_BALANCEADA

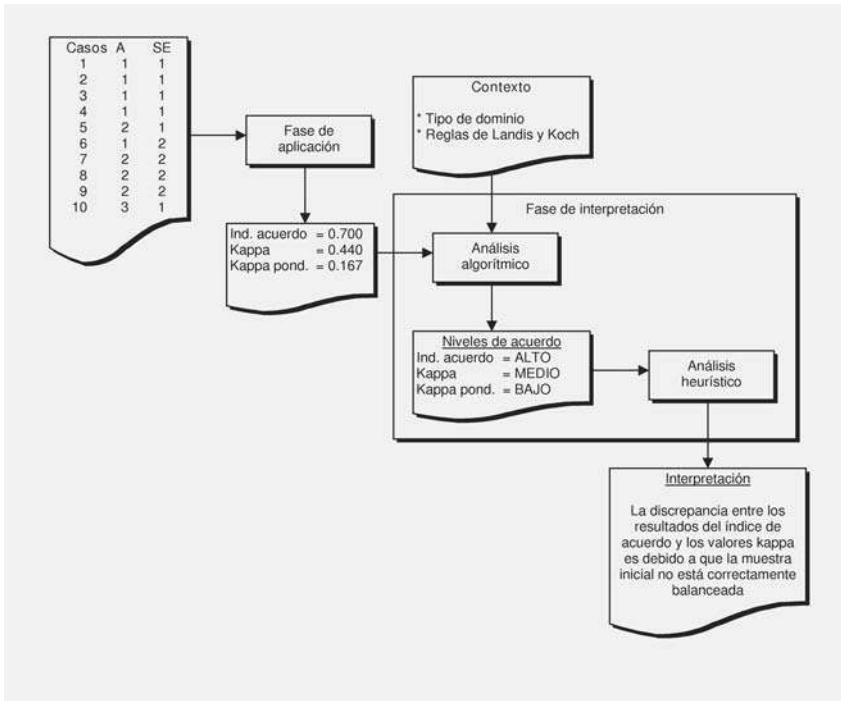


Figura 21.13: Interpretación de resultados de validación.

Es decir, si valores altos del índice de acuerdo coinciden con valores bajos de kappa ponderada, esto quiere decir que la muestra de entrada no está correctamente balanceada sobre todas las posibles categorías en las que se divide la interpretación considerada. Como vemos, el análisis heurístico trata con información ya procesada por el análisis algorítmico, lo que nos permite definir reglas con economía de expresión y cuya sintaxis sea más cercana al lenguaje natural. En IA este conocimiento se denomina de alto nivel.

Si analizamos la tabla de contingencia de los datos de validación de este ejemplo en concreto veremos que, efectivamente, la información de entrada no está correctamente balanceada sobre las distintas categorías. En este caso se pueden realizar sugerencias para modificar los resultados, como hacer un nuevo muestreo que contemple casos de todas las categorías estudiadas.

21.6 Resumen

La fase de verificación y validación de los sistemas inteligentes es, frecuentemente, la gran olvidada dentro de las metodologías de desarrollo, no sólo de este tipo de sistemas, sino que también podría decirse lo mismo del software tradicional.

El principal inconveniente al que se enfrenta la fase de verificación es su dependencia del paradigma de desarrollo utilizado para construir el sistema. No es lo mismo la verificación de un sistema basado en reglas que un sistema basado en redes bayesianas. Además, en el texto hemos visto cómo dentro de un mismo paradigma, pequeñas modificaciones como la inclusión de medidas de incertidumbre pueden variar por completo nuestro esquema de verificación.

La fase de validación no se enfrenta a este problema, ya que trata al sistema como una caja negra, pero se enfrenta a otro de resolución complicada: la necesidad de identificar un criterio de validación, es decir, quién nos indica lo que es una salida válida. En el software tradicional suele ser una respuesta fácil de obtener, si queremos comprobar si un método calcula correctamente la media de una serie de valores no hay más que pasárselas de prueba para los cuales ya hemos precalculado nosotros la salida correcta. Después, simplemente habrá que comprobar la salida obtenida con la salida deseada.

Pero en los sistemas inteligentes las cosas no son tan sencillas, hay que recordar que estamos trabajando en dominios complejos y de naturaleza heurística. Por ello, una de las soluciones más habituales es probar el sistema comparando sus resultados con los de un grupo de expertos del dominio y comprobar si el sistema resulta indistinguible de los expertos humanos, en el sentido de que las discrepancias entre el sistema y los expertos sean similares a las discrepancias que existen entre los propios expertos (lo que sería una actualización del mítico test de Turing).

En este capítulo hemos visto también que, generalmente, no existe una única medida estadística que nos dé la información que estamos buscando. Es más, normalmente necesitaremos la utilización conjunta de varias medidas y su análisis comparativo para poder establecer conclusiones e interpretaciones adecuadas.

Por último, destacar que la validación no es el último paso en el análisis del comportamiento de un sistema inteligente. Quedan otras fases, que en este capítulo hemos denominado genéricamente como evaluación y que incluirían aspectos que van más allá de la corrección de los resultados finales. Por ejemplo, podemos citar la usabilidad, es decir, si el sistema que estamos desarrollando es fácil de usar, flexible, consistente, está bien documentado, etc. También podríamos citar la utilidad, es decir, que el sistema añade valor a lo ya existente en el entorno de trabajo y no provoque cambios sustanciales en los procedimientos habituales que se llevan a cabo. Pero todas estas tareas se escapan del campo de la IAI y los sistemas inteligentes para adentrarse más en el campo de la computación y de la ingeniería del software.

21.7 Ejercicios resueltos

Se ha desarrollado un SBC que trata de clasificar el valor numérico de la presión arterial en una de las siguientes categorías semánticas: Hipertensión Severa (HeS), Hipertensión Ligera (HeL), Normotensión (NoT), Hipotensión Ligera (HoL) e Hipotensión Severa (HoS).

Para validar el SBC desarrollado se han recogido 10 casos de prueba (el número de casos de prueba se ha escogido bajo por motivos de sencillez, ya que sólo 10 casos

serían poco representativos en un dominio real). Estas situaciones de prueba se han presentado, de forma independiente, al SBC y a cuatro expertos humanos del dominio (nombrados como A, B, C y D). Los resultados pueden verse en la Tabla 21.1. Dados estos datos, realizar los siguientes ejercicios:

Experto/Caso	A	B	C	D	SBC
1	NoT	NoT	HeL	HeL	NoT
2	HeL	HeL	HoL	HoL	HeL
3	HeS	HeS	HeL	NoT	HeL
4	HoL	NoT	HoL	HoL	NoT
5	HoS	HoS	HoS	HoS	HoS
6	NoT	NoT	NoT	NoT	NoT
7	NoT	HoL	HeL	HeL	HoS
8	HeL	NoT	NoT	NoT	NoT
9	HoL	NoT	HoL	NoT	HoL
10	HeS	HeL	HeL	NoT	HeS

Tabla 21.1: Tabla de casos de prueba para la evaluación del SBC de ejemplo

21.1. Encontrar las tablas de contingencia para todos los posibles pares de expertos que puedan formarse: A/B, A/C, B/C, etc.

21.2. Calcular los valores del índice de acuerdo y del índice de acuerdo dentro de uno para todos los posibles pares de expertos, incluyendo al SBC.

21.3. Calcular los valores de kappa y kappa ponderada (suponiendo una progresión geométrica de pesos de desacuerdo) para todos los posibles pares de expertos, incluyendo al SBC.

21.4. Basándose en los resultados del índice de acuerdo, calcular los valores del índice de Williams para todos los expertos.

21.5. Basándose en los resultados del índice de acuerdo calcular el resultado del análisis de agrupamientos.

21.6. Supóngase ahora que A es el experto con mayor experiencia en el campo, por lo que se puede considerar como una referencia estándar. También supóngase que nuestro mayor interés es detectar correctamente los casos de hipertensión (sea ligera o severa) ya que son los que más peligro llevan. Dadas estas suposiciones calcular los distintos ratios de acuerdo para el SBC tomando como referencia los datos de A, para la categoría hipertensión (sin diferenciar entre severa y ligera).

21.7. En base a los resultados anteriores, y suponiendo que el número de casos empleado hubiera sido mayor, ¿qué conclusiones se pueden sacar del estudio? ¿se podría considerar que el SBC es válido?

Solución:

21.1 Encontrar las tablas de contingencia

El primer paso para crear una tabla de contingencia es disponer las frecuencias absolutas, tal y como se muestra en la siguiente tabla para los expertos A y B.

A / B	HeS	HeL	NoT	HoL	HoS	
HeS	1	1	0	0	0	2
HeL	0	1	1	0	0	2
NoT	0	0	2	1	0	3
HoL	0	0	2	0	0	2
HoS	0	0	0	0	1	1
	1	2	5	1	1	

Posteriormente, las frecuencias absolutas se convierten en relativas. En esta ocasión al ser sólo 10 casos la operación es sencilla.

A / B	HeS	HeL	NoT	HoL	HoS	
HeS	0,1	0,1	0	0	0	0,2
HeL	0	0,1	0,1	0	0	0,2
NoT	0	0	0,2	0,1	0	0,3
HoL	0	0	0,2	0	0	0,2
HoS	0	0	0	0,0	0,1	0,1
	0,1	0,2	0,5	0,1	0,1	

Actuando igual para los otros pares de expertos obtenemos las siguientes tablas:

A / C	HeS	HeL	NoT	HoL	HoS	
HeS	0	0,2	0	0	0	0,2
HeL	0	0	0,1	0,1	0	0,2
NoT	0	0,2	0,1	0	0	0,3
HoL	0	0	0	0,2	0	0,2
HoS	0	0	0	0	0,1	0,1
	0	0,4	0,3	0,2	0,1	

A / D	HeS	HeL	NoT	HoL	HoS	
HeS	0	0	0,2	0	0	0,2
HeL	0	0	0,1	0,1	0	0,2
NoT	0	0,2	0,1	0	0	0,3
HoL	0	0	0,1	0,1	0	0,2
HoS	0	0	0	0	0,1	0,1
	0	0,2	0,5	0,2	0,1	

A / SE	HeS	HeL	NoT	HoL	HoS	
HeS	0,1	0,1	0	0	0	0,2
HeL	0	0,1	0,1	0	0	0,2
NoT	0	0	0,2	0	0,1	0,3
HoL	0	0	0,1	0,1	0	0,2
HoS	0	0	0	0	0,1	0,1
	0,1	0,2	0,4	0,1	0,2	

B / C	HeS	HeL	NoT	HoL	HoS	
HeS	0	0,1	0	0	0	0,1
HeL	0	0,1	0	0,1	0	0,2
NoT	0	0,1	0,2	0,2	0	0,5
HoL	0	0,1	0	0	0	0,1
HoS	0	0	0	0	0,1	0,1
	0	0,4	0,2	0,3	0,1	

B / D	HeS	HeL	NoT	HoL	HoS	
HeS	0	0	0,1	0	0	0,1
HeL	0	0	0,1	0,1	0	0,2
NoT	0	0,1	0,3	0,1	0	0,5
HoL	0	0,1	0	0	0	0,1
HoS	0	0	0	0	0,1	0,1
	0	0,2	0,5	0,2	0,1	

B / SE	HeS	HeL	NoT	HoL	HoS	
HeS	0	0,1	0	0	0	0,1
HeL	0,1	0,1	0	0	0	0,2
NoT	0	0	0,4	0,1	0	0,5
HoL	0	0	0	0	0,1	0,1
HoS	0	0	0	0	0,1	0,1
	0,1	0,2	0,4	0,1	0,2	

C / D	HeS	HeL	NoT	HoL	HoS		C / SE	HeS	HeL	NoT	HoL	HoS	
HeS	0	0	0	0	0	0	HeS	0	0	0	0	0	0
HeL	0	0,2	0,2	0	0	0,4	HeL	0,1	0,1	0,1	0	0,1	0,4
NoT	0	0	0,2	0	0	0,2	NoT	0	0	0,2	0	0	0,2
HoL	0	0	0,1	0,2	0	0,3	HoL	0	0,1	0,1	0,1	0	0,3
HoS	0	0	0	0,1	0,1	0,1	HoS	0	0	0	0	0,1	0,1
	0	0,2	0,5	0,2	0,1			0,1	0,2	0,4	0,1	0,1	0,2

D / SE	HeS	HeL	NoT	HoL	HoS	
HeS	0	0	0	0	0	0
HeL	0	0	0,1	0	0,1	0,2
NoT	0,1	0,1	0,2	0,1	0	0,5
HoL	0	0,1	0,1	0	0	0,2
HoS	0	0	0	0	0,1	0,1
	0,1	0,2	0,4	0,1	0,2	

21.2 Cálculo del índice de acuerdo y el índice de acuerdo dentro de uno

Si tomamos la matriz de contingencia entre A y B resulta sencillo calcular el índice de acuerdo, simplemente hay que sumar los valores de la diagonal principal:

$$\text{Índice Acuerdo(A,B)} = 0,1 + 0,1 + 0,2 + 0 + 0,1 = 0,5$$

Para el índice de acuerdo dentro de uno hay que sumar además las diagonales adyacentes a la diagonal principal en la tabla de contingencia.

$$\text{Índice Acuerdo Uno(A,B)} = 0,5 + (0,1 + 0,1 + 0) + (0 + 0 + 0,2 + 0) = 0,9$$

Actuando de la misma forma para el resto de pares de expertos obtenemos las siguientes tablas resumen:

Acuerdo	A	B	C	D	SE	Ac. en uno	A	B	C	D	SE
A	-	0,5	0,4	0,3	0,6	A	-	1	0,9	0,7	0,9
B	0,5	-	0,4	0,4	0,6	B	1	-	0,8	0,7	1
C	0,4	0,4	-	0,7	0,5	C	0,9	0,8	-	1	0,8
D	0,3	0,4	0,7	-	0,3	D	0,7	0,7	1	-	0,7
SE	0,6	0,6	0,5	0,3	-	SE	0,9	1	0,8	0,7	-

21.3 Cálculo de kappa y kappa ponderada

Para el cálculo de kappa necesitamos calcular el porcentaje de acuerdo debido a la casualidad de cada casilla, esto se hace multiplicando las proporciones marginales pertenecientes a dicha casilla. Por ejemplo, para el par de expertos A/B las tablas correspondientes a las proporciones de acuerdo observadas (p_o) y a las proporciones de acuerdo debidas a la casualidad (p_c) son:

A / B (p_o)	HeS	HeL	NoT	HoL	HoS	
HeS	0,1	0,1	0	0	0	0,2
HeL	0	0,1	0,1	0	0	0,2
NoT	0	0	0,2	0,1	0	0,3
HoL	0	0	0,2	0	0	0,2
HoS	0	0	0	0,0	0,1	0,1
	0,1	0,2	0,5	0,1	0,1	

A / B (p_c)	HeS	HeL	NoT	HoL	HoS	
HeS	0,02	0,04	0,1	0,02	0,02	0,2
HeL	0,02	0,04	0,1	0,02	0,02	0,2
NoT	0,03	0,06	0,15	0,03	0,03	0,3
HoL	0,02	0,04	0,1	0,02	0,02	0,2
HoS	0,01	0,02	0,05	0,01	0,01	0,1
	0,1	0,2	0,5	0,1	0,1	

Por ejemplo, es fácil comprobar que el valor debido a la casualidad para la casilla superior izquierda (HeS, HeS) es 0,02, que es igual a la multiplicación de sus marginales ($0,1 \times 0,2$). En base a estas dos matrices podemos calcular el valor de kappa como sigue:

$$p_o = 0,1 + 0,1 + 0,2 + 0 + 0,1 = 0,5$$

$$p_c = 0,02 + 0,04 + 0,15 + 0,02 + 0,01 = 0,24$$

$$\kappa = \frac{p_o - p_c}{1 - p_c} = \frac{0,5 - 0,24}{1 - 0,24} = \frac{0,26}{0,76} = 0,342$$

Para el valor de kappa ponderada necesitamos conocer en primer lugar qué pesos de desacuerdo vamos a utilizar. El enunciado habla de pesos que sigan una progresión geométrica de la siguiente forma:

A / B (pesos)	HeS	HeL	NoT	HoL	HoS
HeS	0	1	4	9	16
HeL	1	0	1	4	9
NoT	4	1	0	1	4
HoL	9	4	1	0	1
HoS	16	9	4	1	0

Con este tipo de pesos intentamos penalizar mucho los desacuerdos graves, pero muy poco los desacuerdos que pueden ser debidos a cuestiones de matiz lingüístico.

Para calcular kappa ponderada calculamos en primer lugar los siguientes valores del porcentaje de acuerdo observado y el porcentaje de acuerdo debido a la casualidad, pero para todas las celdas y multiplicado por su correspondiente peso de desacuerdo:

$$\sum_{i=1}^k v_{ij} p_{oij} = (0,1 \times 0) + (0,1 \times 1) + (0 \times 4) + (0 \times 9) + (0 \times 16) + (0 \times 1) + \dots + (0,1 \times 0) = 0,5$$

$$\sum_{i=1j=1}^k v_{ij} p_{cij} = (0,02 \times 0) + (0,04 \times 1) + (0,1 \times 4) + (0,02 \times 9) + \\ + (0,02 \times 16) + \dots + (0,01 \times 0) = 2,66$$

Podemos ahora calcular kappa ponderada como:

$$\kappa_w = 1 - \frac{\sum_{i=1j=1}^k v_{ij} p_{oij}}{\sum_{i=1j=1}^k v_{ij} p_{cij}} = 1 - \frac{0,5}{2,66} = 1 - 0,188 = 0,812$$

Como se ve, el valor de kappa ponderada (0,812) ha mejorado sustancialmente el valor de kappa (0,342), debido a que todas las discrepancias se sitúan en las diagonales adyacentes a la principal en la matriz de contingencia.

Actuando de la misma forma para el resto de pares de expertos, obtenemos las siguientes tablas resumen:

k	A	B	C	D	SE
A	-	0,342	0,241	0,079	0,487
B	0,342	-	0,231	0,118	0,444
C	0,241	0,231	-	0,600	0,367
D	0,079	0,118	0,600	-	0,028
SE	0,487	0,444	0,367	0,028	-

k_w	A	B	C	D	SE
A	-	0,812	0,672	0,355	0,777
B	0,812	-	0,459	0,227	0,847
C	0,672	0,459	-	0,839	0,380
D	0,355	0,227	0,839	-	0,071
SE	0,777	0,847	0,380	0,071	-

21.4 Cálculo del índice de Williams para el índice de acuerdo

En primer lugar, partimos ahora de la matriz resumen para el índice de acuerdo que hemos calculado previamente:

Acuerdo	A	B	C	D	SE
A	-	0,5	0,4	0,3	0,6
B	0,5	-	0,4	0,4	0,6
C	0,4	0,4	-	0,7	0,5
D	0,3	0,4	0,7	-	0,3
SE	0,6	0,6	0,5	0,3	-

El índice de Williams para el SBC se calcularía como sigue: el acuerdo dentro del grupo (P_n) es la media de acuerdos entre expertos que no son el SBC:

$$P_n = \frac{0,5 + 0,4 + 0,3 + 0,4 + 0,4 + 0,7}{6} = \frac{2,7}{6} = 0,45$$

El acuerdo del SBC con el grupo de referencia se calcula como la media de los acuerdos entre el SBC y el resto de expertos:

$$P_o = \frac{0,6 + 0,6 + 0,5 + 0,3}{4} = \frac{2}{4} = 0,5$$

De esta forma, podemos definir el índice de Williams como:

$$I_o = \frac{0,5}{0,45} = 1,111$$

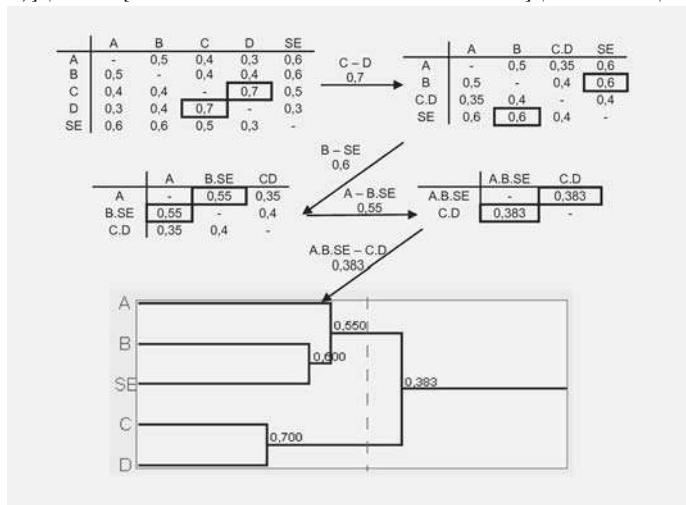
Obtener un valor superior a 1 indica que el SBC no desentonaría con el resto de expertos si se encontrase dentro del mismo grupo.

21.5 Cálculo del análisis de agrupamientos para el índice de acuerdo

Partiendo de la misma matriz resumen que en el caso anterior, el cálculo del análisis de agrupamientos es sencillo siguiendo el algoritmo SAHN explicado previamente. En primer lugar, buscamos el máximo valor en la matriz (en la primera iteración es el valor 0,7 correspondiente a los expertos C y D) y posteriormente unimos dichos expertos en un grupo. En este caso, definimos la distancia del nuevo grupo formado a los otros grupos como la media de las distancias individuales. Así, tenemos $d(A, C) = 0,4$, $d(A, D) = 0,3$, por lo tanto $d(A, C.D) = 0,35$.

En el segundo paso de la unión vemos que existen dos valores máximos iguales: $d(A, SE) = 0,6$ y $d(B, SE) = 0,6$. Es un caso poco común pero que obliga a tomar una decisión arbitraria de unir un par de expertos primero, lo que puede desvirtuar en cierta medida el resultado final.

En la unión del último grupo puede verse cómo el resultado final es el resultado de las distancias medias entre los elementos de los clústeres que se unen: $d(A.B.SE, C.D) = [d(A, C) + d(A, D) + d(B, C) + d(B, D) + d(SE, C) + d(SE, D)] / 6 = [0,4 + 0,3 + 0,4 + 0,4 + 0,5 + 0,3] / 6 = 2,3 / 6 = 0,383$



21.6 Cálculo de los ratios de acuerdo suponiendo a A como referencia para la unión de categorías HeS y HeL

Para calcular los ratios de acuerdo debemos, en primer lugar, calcular la tabla de contingencia 2×2 que va a relacionar los resultados del SBC con los resultados del experto que actúa como referencia (A). Recordemos que no vamos a suponer diferencias entre las categorías de hipertensión ligera e hipertensión severa. La tabla resultante sería:

		A		3
		Hiper	\neg Hiper	
SE	Hiper	$a = 3$	$b = 0$	
	\neg Hiper	$c = 1$	$d = 6$	7
		4	6	

A partir de esta tabla resulta sencillo el cálculo de los ratios de acuerdo:

$$\text{Sensibilidad} = \frac{a}{a + c} = \frac{3}{4} = 0,75$$

$$\text{Especificidad} = \frac{d}{b + d} = \frac{6}{6} = 1$$

$$\text{Valor Predictivo Positivo} = \frac{a}{a + b} = \frac{3}{3} = 1$$

$$\text{Valor Predictivo Negativo} = \frac{d}{c + d} = \frac{6}{7} = 0,857$$

$$\text{Área bajo ROC} = \frac{\text{Sen.} + \text{Esp.}}{2} = \frac{0,75 + 1}{2} = \frac{1,75}{2} = 0,875$$

21.7 Conclusiones

De los resultados parece desprenderse que existen dos grupos de expertos, uno formado por los expertos C y D, y otro formado por los expertos A, B y el SBC. Los grupos se mantienen en cualquiera de las cuatro medidas de pares consideradas, lo que nos da una idea de su homogeneidad. Los resultados de las medidas de grupo permiten visualizar gráficamente estos grupos, tal y como se aprecia en los resultados del análisis de agrupamientos. Podemos también concluir que el SBC es indistinguible de los demás expertos del dominio, por lo que en principio podemos aceptarlo como válido, aunque esto último depende en gran medida del contexto en el que nos estamos moviendo y de los objetivos que se pretenden alcanzar. Lo adecuado sería que, a la vista de estos resultados, se realizara una nueva reunión con los expertos para intentar analizar las discrepancias obtenidas.

En cuanto a los ratios de acuerdo podemos deducir que la capacidad del SBC para detectar casos de hipertensión es muy buena. Tanto los casos positivos

(considerados como los que presentan hipertensión), como los casos negativos (sin hipertensión). Quizá el valor más bajo es el de la sensibilidad, es decir, no se han detectado todos los casos de hipertensión posible, pero esto se compensa al analizar el valor predictivo positivo, todos los casos de hipertensión detectados son correctos, no hay falsos positivos.

21.8 Ejercicios propuestos

21.1. Tenemos la siguiente base de reglas que nos pretende asesorar si es adecuado ir a jugar un partido de fútbol a la playa. Suponiendo un razonamiento encadenado hacia adelante partimos de los siguientes datos: podemos consultar el tiempo asomándonos por la ventana, llamar a los amigos a ver si quieren venir, y podemos consultar la tabla de mareas de la zona. Según estos datos, ¿qué anomalías hay en dicha base de reglas? ¿cambian estas anomalías cuando el encadenamiento es regresivo sobre la hipótesis “ir a jugar”?

REGLAS:

SI hay amigos Y el día es soleado ENTONCES ir a jugar

SI el día es lluvioso ENTONCES no ir a jugar

SI no hay amigos y la marea es alta ENTONCES no ir a jugar

SI el día es lluvioso Y no hay amigos ENTONCES no ir a jugar

SI no hay pelota ENTONCES no ir a jugar

SI no hay amigos Y la marea es baja ENTONCES no ir a jugar

SI el día es soleado Y hay amigos ENTONCES ir a jugar

SI hay amigos ENTONCES buscar pelota para jugar

21.2. Durante un experimento de validación se analizaron un total de 10 casos, de manera ciega e independiente, por dos expertos (A,B), y por un SBC (SE). Los resultados obtenidos fueron los siguientes:

Experto/Caso	A	B	SE
1	Muy negativo	Indiferente	Bastante negativo
2	Algo positivo	Algo positivo	Bastante positivo
3	Algo negativo	Algo negativo	Algo negativo
4	Bastante positivo	Bastante positivo	Muy negativo
5	Muy positivo	Bastante negativo	Muy positivo
6	Algo negativo	Muy positivo	Indiferente
7	Indiferente	Muy negativo	Indiferente
8	Bastante negativo	Indiferente	Algo negativo
9	Bastante positivo	Algo positivo	Muy positivo
10	Indiferente	Algo negativo	Indiferente

Donde los valores de la tabla son etiquetas que forman parte de una escala semántica, lingüisticamente ordenada de la siguiente forma: Muy Positivo, Bastante Positivo,

Algo Positivo, Indiferente, Algo Negativo, Bastante Negativo, Muy Negativo. Se pide la resolución de las siguientes cuestiones:

- Encontrar las tablas de contingencia A/B, A/SE, B/SE.
- Calcular los valores del índice de acuerdo y del índice de acuerdo dentro de uno.
- Calcular los valores de kappa y kappa ponderada. ¿qué pesos de desacuerdo se considerarían más adecuados?
- Calcular los valores del índice de Williams para todos los expertos.
- Basándose en los resultados del índice de acuerdo, calcular el resultado del análisis de agrupamientos.
- Comentar los resultados obtenidos.

21.3. Se han desarrollado dos sistemas que permiten clasificar un animal en una categoría dada (mamífero, ave, reptil, pez, etc.) según sus características (tiene pelo, tiene plumas, es ovíparo o vivíparo, etc.). Los resultados de los dos sistemas se resumen en las siguientes matrices de confusión, que muestran la comparación entre la clasificación del sistema y la clasificación real del animal. El índice de acuerdo de ambos sistemas es muy similar: sistema #1 = 0,96 y sistema #2 = 0,921. Utilizar los ratios de acuerdo para analizar los resultados por categorías y decidir qué sistema es el mejor clasificador.

Clasificación hecha por el sistema #1							Clasificación real
mamífero	ave	reptil	pez	anfibio	insecto	invertebrado	
41	0	0	0	0	0	0	
0	20	0	0	0	0	0	
0	1	3	1	0	0	0	
0	0	0	13	0	0	0	
0	0	1	0	3	0	0	
0	0	0	0	0	8	0	
0	0	0	0	0	1	9	

Clasificación hecha por el sistema #2						
mamífero	ave	reptil	pez	anfibio	insecto	invertebrado
41	0	0	0	0	0	0
0	20	0	0	0	0	0
0	0	3	1	0	1	0
0	0	0	13	0	0	0
0	0	1	0	3	0	0
0	0	0	0	0	5	3
0	0	0	0	0	2	8

21.4. Después de realizar la validación de un sistema nos encontramos con la siguiente tabla resumen para el índice de acuerdo:

	A	B	C	D	E	F	SBC
A	...	0,448	0,448	0,724	0,586	0,621	0,414
B	0,448	...	0,138	0,517	0,724	0,655	0,552
C	0,448	0,138	...	0,414	0,310	0,241	0,379
D	0,724	0,517	0,414	...	0,655	0,483	0,552
E	0,586	0,724	0,310	0,655	...	0,655	0,517
F	0,621	0,655	0,241	0,483	0,655	...	0,621
SBC	0,414	0,552	0,379	0,552	0,517	0,621	...

Si calculamos el índice de Williams para el SBC obtenemos un valor de 0,995. Es decir, un valor cercano al 1 que parece indicar que el SBC se integra perfectamente dentro del grupo de expertos. Sin embargo ¿puede realizarse esta conclusión sin lugar a dudas? Calcular la medida de Williams para el resto de expertos ¿qué opinión merece el resultado del experto C?

Referencias

- BOEHM, B.W.: *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- CHANDRASEKARAN, B.: «On Evaluating AI Systems for Medical Diagnosis». *AI Magazine*, 1983, pp. 4(2), 34–37.
- COHEN, J.: «A coefficient of agreement for nominal scales». *Educational and Psychological Measurement*, 1960, pp. 20, 37–46.
- COHEN, J.: «Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit». *Psychological Bulletin*, 1968, pp. 70(4), 213–220.
- DUBES, R.C.: «Cluster analysis and related issues». En: *Handbook of Pattern Recognition and Computer Vision*, pp. 3–32. C.H. Chen, L.F. Pau and P.S.P. Wang (eds.), World Scientific Publishing Company, 1993.
- FAMILI, A.; SHEN, W.M.; WEBER, R. y SIMOUDIS, E.: «Data Pre-processing and Intelligent Data Analysis». *International Journal on Intelligent Data Analysis*, 1996, **1**(1), pp. 1–28.
- FLEISS, J.L.: «Measuring nominal scale agreement among many raters». *Psychological Bulletin*, 1971, pp. 76(5), 378–382.
- GONZÁLEZ, A.J. y DANKEL, D.D.: *The Engineering of Knowledge-Based Systems: Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993.
- GUPTA, U.: «Validation and Verification of Knowledge-Based Systems: A Survey». *Journal of Applied Intelligence*, 1993, pp. 3, 343–363.
- HOPPE, T. y MESEGURER, P.: «VVT Terminology: A Proposal». *IEEE Expert*, 1993, pp. June, 48–55.
- JACCARD, P.: «Nouvelles recherches sur la distribution florale». *Bull. Soc. Vaud. Sci. Nat.*, 1908, **44**, pp. 223–270.
- LAMBERTI, D.M. y NEWSOME, S.L.: «Presenting abstract versus concrete information in expert systems: What is the impact on user performance?» *International Journal of Man-Machine Studies*, 1989, pp. 31(7), 27–45.
- LIGHT, R.J.: «Measures of response agreement for qualitative data: some generalizations and alternatives». *Psychological Bulletin*, 1971, pp. 76, 365–377.
- LÓPEZ, B.; MESEGURER, P. y PLAZA, E.: «Knowledge Based Systems Validation: A State of the Art». *AICOM*, 1990, pp. 3(2), June.
- MOSQUEIRA-REY, E. y MORET-BONILLO, V.: «Validation of Intelligent Systems: A Critical Study and a Tool». *Expert Systems with Applications*, 2000, pp. 18(1), 1–16.

- NGUYEN, T.A.; WALTON, A.P.; LAFFEY, T.J. y PECORA, D.: «Knowledge Base Verification». *AI Magazine*, 1987, pp. 8(2), 69–75.
- O'KEEFE, R.M.; BALCI, O. y SMITH, E.P.: «Validating Expert System Performance». *IEEE Expert*, 1987, pp. 2(4), 81–89.
- O'LEARY, D.E.: «Verifying and Validating Expert Systems: A Survey». En: *Expert Systems in Business and Finance: Issues and Applications*, pp. 181–208. P.R. Watkins and L.B. Eliot (Eds.), John Wiley & Sons Ltd, 1993.
- SACKMAN, H.: *Delphi Assessment : Expert Opinion, Forecasting and Group Process*. Santa Monica: The Rand Corporation, 1974.
- WILLIAMS, G.W.: «Comparing the joint agreement of several raters with another rater». *Biometrics*, 1976, pp. 32, 619–627.

Capítulo 22

Razonamiento basado en casos

José Manuel Juárez Herrero y José Tomás Palma Méndez
Universidad de Murcia

22.1 Introducción

En los capítulos anteriores, se han analizado distintas formas de construir sistemas inteligentes que, de algún modo, simulan la manera en la que los seres humanos resuelven problemas. Por ejemplo, los Sistemas Basados en Reglas (3) se apoyan en la existencia de una base de conocimiento, formada por un conjunto de reglas que de alguna forma *codifican la experiencia humana*. Una vez que disponemos de una base de conocimiento para un dominio concreto, sólo tenemos que plantear al sistema el problema que queremos resolver. El motor de inferencia empezará a aplicar reglas mientras sea posible. El proceso terminará cuando el sistema haya alcanzado una solución satisfactoria, o bien informará que no puede resolver el problema. Como se puede ver esta aproximación, es bastante similar a la forma en la que un humano se enfrenta a un juego como el ajedrez. Cada jugador se enfrenta a su oponente desarrollando una estrategia y ateniéndose a las reglas del juego en cada paso, realizando los movimientos que más le convengan. Así, en un sistema basado en reglas no sólo nos encontramos con las reglas que indican cuáles son los posibles movimientos, sino que también hay reglas que representan las estrategias más convenientes como resultado de la experiencia.

Sin embargo, resulta simplista considerar que el cerebro humano resuelve los problemas de esta forma y siempre partiendo de cero. Generalmente, cuando intentamos resolver un problema ya resuelto es más sencillo, ya que podemos basarnos en la experiencia. Tareas que utilizan esta experiencia son, por ejemplo, la que realiza un juez, un cocinero o los ejemplos de jugadas de un libro de ajedrez. Estas situaciones fueron las que llevaron a Schank y Abelson [Schank y Abelson, 1977] a plantear su teoría SPGU: Scripts, Plans, Goals and Understanding (guiones, planes, objetivos y comprensión). El objetivo inicial de Schank y Abelson consistía en explicar el proceso mediante el cual era posible entender párrafos en lenguaje natural. La conclusión a la que llegaron es que la comprensión de textos está dirigida por esquemas mentales,

que de alguna forma permiten llenar los huecos que genera lo que no está escrito. De esta forma, la comprensión de textos se puede ver como una forma de explicación de lo que se está leyendo según dichos esquemas mentales. Schank propuso que dichos esquemas mentales se pueden representar mediante guiones (del inglés *script*) que, en cierto modo, detallan cada una de las situaciones y que, a su vez, pueden ser detalladas mediante otros guiones. Por lo tanto, un guión representa un determinado episodio en el contexto de un comportamiento determinado, definido mediante un conjunto de eventos ordenados, tal y como se espera que ocurran. Cuando nos enfrentamos a una nueva situación, el guión es adaptado a dicha situación para cubrir con sus excepciones y completarla con información útil. Para que esto sea posible, un guión debe contener:

- Los objetivos reales.
- Los planes que llevan a la consecución de dichos objetivos.
- Los papeles que desarrollan los individuos involucrados y su personalidad.
- Un contexto social.
- Las variaciones específicas del patrón representado.
- Elementos que participan en el patrón.
- Propiedades, precondiciones y postcondiciones.

La utilización de este tipo de guiones requiere de un proceso compuesto de dos pasos: (1) recuperación del guión que mejor se adapte a la situación actual, y (2) aplicación del mismo a la situación actual. Estas consideraciones llevaron a Schank a plantear una nueva teoría: la *Memoria Dinámica* [Schank, 1982]. En esta nueva propuesta, el proceso de comprensión intenta recordar situaciones pasadas que se parezcan a la situación actual. Estas situaciones pasadas asisten al proceso de comprensión y permiten elaborar una respuesta adecuada. Para que esto sea posible, la memoria humana tiene que estar compuesta por experiencias o episodios pasados, que son almacenadas como generalizaciones, y que por lo tanto, admiten su aplicación en otros contextos. Son estas generalizaciones las que proporcionan el guión adecuado para su búsqueda, suministrando índices a distintas partes de los episodios para encontrarlos fácilmente. Esta teoría fue la que se basó Janet Kolodner para desarrollar el primer sistema de Razonamiento Basado en Casos (SRBC), CYRUS [Kolodner y Leake, 1996], y cuyo objetivo era dar respuesta a consultas relativas a viajes y citas para una secretaría concreta de la administración pública estadounidense. Durante los años 80 aumentó el interés en este campo, inicialmente en Estados Unidos y posteriormente en Europa, incrementando el número de grupos centrados en la investigación en SRBCs. Por ejemplo, durante este periodo cabe destacar el trabajo llevado a cabo por el grupo liderado por Bruce W. Porter de la Universidad de Texas en el desarrollo del sistema PROTOS, un SRBC para el diagnóstico médico de afecciones del oído [Bareiss, 1988]. Otros muchos sistemas fueron también propuestos como: CASEY,

que diagnósticaba enfermedades cardíacos [Koton, 1989]; KRITIK, para el ensamblaje físico de componentes [Goel, 1989]; MEDIATOR [Simpson, 1985], PERSUADER [Sycara, 1987]; CHEF [Hammond, 1986], sistema para planificar recetas de cocina; JULIA [Hinrichs. y otros, 1991], sistema que diseñaba menús en restaurantes, etc. A partir de la década de los 90, los SRBCs han sido incorporados gradualmente a la industria en diferentes sectores (automoción, industria plástica, etc.) para resolver un gran número de problemas tales como el diagnóstico, la planificación o la toma de decisiones. Este gran interés se apoya fundamentalmente en las grandes bases de datos corporativas. En la actualidad, la mayor parte de las grandes empresas poseen sistemas de información corporativos. Estos sistemas han ido almacenando información sobre el funcionamiento de la empresa durante años, convirtiéndose en grandes repositorios donde se recoge la experiencia de la compañía. Es en este contexto donde el desarrollo de SRBCs está viviendo una segunda juventud. Por ejemplo, dos de los campos donde estos sistemas están resultando muy efectivos son: los servicios de ayuda en línea y la sanidad. Muchas de las empresas que ofrecen servicios de ayuda en línea (por ejemplo, las empresas que ofrecen servicios de internet) poseen sistemas RBC que explotan bases de problemas resueltos con el fin de dar con la solución a nuevos problemas que plantean los usuarios. En el campo de la medicina es evidente, ¿qué mejor servicio de diagnóstico que aquél que se puede basar en la experiencia recogida durante largos años de práctica médica?

22.2 Sistemas de Razonamiento Basado en Casos

Básicamente, el RBC es un modelo de razonamiento que nos permite resolver problemas, entender situaciones y aprender, todo ello integrado con los mecanismos asociados a la memoria. Este modelo nos ofrece un nuevo paradigma para la construcción de sistemas inteligentes que se basa en la utilización de la experiencia previa [Kolodner y Leake, 1996]. De esta forma, podemos construir razonadores que resuelven nuevos problemas mediante la adaptación de soluciones que han sido utilizadas para resolver problemas pasados [Riesbeck y Schank, 1989]. En definitiva, un SRBC almacena un conjunto de problemas ya resueltos y recibe como entrada un nuevo problema. El sistema intentará resolver el problema de entrada buscando, de entre los problemas resueltos, el más semejante y adaptando su solución al problema de entrada. A diferencia de otras aproximaciones de la IA, el RBC no depende únicamente del conocimiento general del dominio del problema, sino que utiliza el conocimiento específico de casos pasados para reutilizarlo como solución del nuevo problema. De esta forma, se evita el cuello de botella que supone la adquisición de conocimiento para el desarrollo de SBC.

¿Qué ventajas ofrecen los SRBCs respecto a los sistemas tradicionales de resolución de problemas? Según David B. Leake [Leake, 1996] existen cinco ventajas frente a los sistemas expertos tradicionales:

- **Adquisición de Conocimiento.** Es bien sabido el cuello de botella que supone el proceso de adquisición de conocimiento en los SBC tradicionales. El proceso de extracción de reglas a partir de un experto requiere un enorme esfuerzo, que

no garantiza la validez de dichas reglas. Los SRBC no requieren este proceso ya que utilizan experiencias previas para resolver casos. En los dominios donde la resolución de problemas se basa en experiencias previas (por ejemplo, ciertos campos de la medicina y la jurisprudencia), el coste de la adquisición de conocimiento para un SRBC es mínimo. Sin embargo, esto no es aplicable a todos los dominios, ya que los casos pueden no estar disponibles, ser difíciles de usar o estar incompletos. En estas situaciones, el desarrollo de un SRBC depende de lo costoso que sea la *ingeniería de los casos*: determinar la información que deben incluir, elegir la representación adecuada y extraer dicha información a partir de los datos disponibles.

- **Mantenimiento del Conocimiento.** En los SBC tradicionales el mantenimiento de la base de conocimiento (BC) es una de las tareas más costosa del proceso de desarrollo (aparte del ya mencionado cuello de botella de la adquisición de conocimiento). Normalmente, el contenido inicial de la BC se ve sujeto a adición del nuevo conocimiento, que puede implicar la redefinición del conocimiento existente, pudiéndose darse el caso de que este llegue a estar obsoleto. Los SRBC permiten que los usuarios añadan nuevos casos a la librería de casos sin intervención del experto (aunque tiene que existir algún mecanismo que valide los nuevos casos). Además, gracias a que los SRBC realizan un aprendizaje incremental, inicialmente pueden empezar a trabajar con un conjunto limitado de casos e ir incorporando nuevos casos a medida que sea necesario.
- **Eficiencia en la Resolución de Problemas.** La reutilización de casos previos permite que se puedan resolver problemas similares sin tener que rehacer el proceso de razonamiento. Además, debido a que los SRBC pueden almacenar casos fallidos, estos pueden servir como avisos para futuros problemas a evitar.
- **Calidad de la Solución.** Existen situaciones en las que las BC de los SBC tradicionales pueden tener reglas imperfectas, especialmente si no se ha entendido correctamente el dominio de aplicación. En estos casos, las soluciones proporcionadas por los casos son más precisas, ya que reflejan qué es lo que realmente ha sucedido en un contexto determinado.
- **Aceptación del Usuario.** Para que un sistema inteligente sea aceptado por los usuarios, éstos deben confiar en las soluciones que aportan. Para ello, resulta extremadamente importante que los usuarios confíen en el proceso que lleva a dichas soluciones. En otras aproximaciones esto resulta tremadamente difícil, como en las redes neuronales 15 y los sistemas basados en reglas 3, ya que pueden resultar incomprensibles para los usuarios. En los SRBC las soluciones están basadas en experiencias previas que pueden ser presentadas a los usuarios para apoyar las conclusiones que alcanza el sistema.

Como es bien sabido, una gran parte de la calidad de las soluciones que pueden ofrecer los SBCs reside en un profundo análisis del dominio de aplicación y del grado de complejidad del problema. En este sentido, los SRBC no son ninguna excepción y

sólo son eficientes en determinadas circunstancias. Así, los SRBC se suelen aplicar en dominios donde:

- No es posible establecer un modelo del dominio, ya que éste está poco estructurado o es parcialmente desconocido. En estas circunstancias, el desarrollo de SBCs convencionales se complica enormemente, ya que el problema asociado al cuello de botella en la adquisición de conocimiento se ve incrementado.
- Los expertos tienen dificultades para explicar (verbalizar) cómo se resuelve un problema. En estas situaciones, el conocimiento asociado a los métodos de resolución de problemas es muy complejo de extraer y formalizar, y añade más complicaciones a la fase de adquisición de conocimiento.

Sin embargo, aparte de las evidentes ventajas de las técnicas de RBC sobre los SBCs convencionales, éstas no son la panacea que permite desarrollar sistemas inteligentes en cualquier dominio. Al igual que otras aproximaciones, para que una solución basada en RBC sea eficiente se tienen que cumplir las siguientes asunciones:

- Los problemas similares tienen soluciones similares, pequeñas variaciones en la definición del problema producen pequeñas variaciones en la solución.
- El dominio de aplicación es regular, es decir, lo que hoy es cierto, mañana también lo es.
- Los problemas son recurrentes, es decir, los problemas solucionados suelen presentarse de nuevo al cabo del tiempo. Si esto no es cierto, no es necesario recordar el problema y su solución.

Teniendo en cuenta las asunciones anteriores, resulta evidente que los SRBC pueden resultar eficientes en áreas como la clasificación, diagnóstico y predicción. Sin embargo, en tareas de síntesis, como la planificación, el diseño por configuración y temporalización, su aplicación resulta más compleja. En las siguientes secciones analizaremos la estructura de los SRBC.

22.3 Elementos de un SRBC

Como ya se dijo, los SRBCs reciben como entrada un problema a resolver (es decir, un caso no resuelto) y el sistema busca el caso más similar para reutilizar su solución con el caso de entrada. Una de las tareas más habituales en este tipo de sistemas es la búsqueda de un caso similar en la Librería de Casos. Una vez recuperados un conjunto de casos, una de las cuestiones más importantes y difíciles de resolver es decidir cuáles son los criterios de similitud entre casos para establecer qué caso recuperado es el que se parece más al caso de entrada. Por lo tanto, podemos entender el proceso de razonamiento en SRBCs como un proceso cíclico en el que se resuelve un problema, se aprende de la experiencia, se resuelve otro problema, y así sucesivamente. En definitiva, para que este ciclo sea posible, un SRBC debe incluir estos elementos:

- Los *Casos*, que constituyen el elemento principal.
- Una *Librería de Casos (LC)* (o alternativamente Memoria de Casos), es decir un almacén donde se recojan las experiencias pasadas y que puedan ser aplicadas a problemas nuevos.
- Un *Esquema de Indexación*. Obviamente los casos deben ser almacenados en la LC de forma organizada. Esta organización debe permitir el acceso eficiente a los casos, de tal forma que el proceso de búsqueda de casos similares a una nueva situación sea lo más eficiente posible.
- Un *Mecanismo de Adaptación* que haga posible la aplicación de los casos almacenados a una situación concreta para, de esta forma, ofrecer soluciones a nuevas situaciones
- Un *Mecanismo de Aprendizaje* que permita determinar cuándo un caso resuelto debe ser incluido en la LC y así permitir que sea utilizado en la resolución de nuevas situaciones. Evidentemente, en la mayoría de las ocasiones, esta acción implica una reestructuración de los índices.

En las siguientes secciones iremos analizando con mayor profundidad cada uno de estos elementos.

22.3.1 Los casos y su descripción

El principal elemento de un SRBC es el *caso*. Por *caso* entendemos una situación concreta que describe un problema. Se corresponde con una situación previamente experimentada, capturada y aprendida, de tal forma que pueda ser reutilizada en la resolución de problemas futuros [Aamodt y Plaza, 1994]. También se puede definir como una porción de conocimiento contextualizado que representa una experiencia que enseña una importante lección [Kolodner y Leake, 1996]. De igual modo, se entiende por un caso nuevo o un caso no resuelto, aquél que describe un nuevo problema que necesita una solución. De todas formas, hay que tener en cuenta que el conocimiento capturado en un caso debe ser operacionalizado para que pueda ser usado computacionalmente y, por lo tanto, debe representar una solución a un problema en un contexto determinado. Esto nos lleva a que una de las primeras tareas que hay que afrontar es la de qué se va a incluir en la representación de un caso y cómo se va a almacenar. La LC es la estructura que nos permite una organización estructurada de los casos pasados con sus soluciones. Establecer con exactitud qué es un caso depende del problema que se desea resolver, las características particulares del dominio concreto de aplicación y las técnicas de resolución que se apliquen. Sin embargo, según [Kolodner y Leake, 1996] un caso debe contener la siguiente información:

- Cómo alcanzar uno o varios objetivos.
- En qué situaciones un conjunto de acciones nos llevan a alcanzar el objetivo.
- Los efectos de cada acción.

Según esta definición, un caso contiene dos partes funcionales: 1) la lección que enseña y 2) el contexto donde puede ser explicada dicha lección, es decir, información que nos sirve para establecer cuándo un caso se recupera para ser utilizado y aplicado a la situación actual. De forma práctica, se suele considerar la estructura de un caso desde el punto de vista conceptual, según la cual un caso está compuesto por:

- La descripción del problema.
- La solución de dicho problema.

Desde un punto de vista más formal, un caso puede verse como una correspondencia entre elementos del espacio de los posibles problemas en elementos del espacio de las posibles soluciones (véase la Figura 22.1). Es decir, podemos definir los casos como una correspondencia entre el conjunto de problemas y el conjunto de posibles soluciones:

$$\text{Caso} = \text{Problema} \times \text{Solución} \quad (22.1)$$

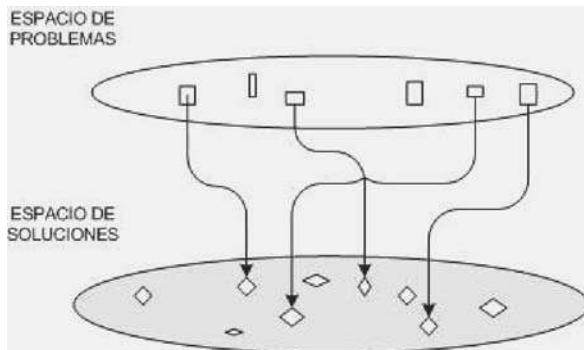


Figura 22.1: Descripción del caso.

A su vez, un problema está compuesto por un conjunto de atributos que determinan los valores de las diferentes características que definen el problema. En la mayoría de las ocasiones, los problemas que describen los casos están compuestos por un conjunto finito de atributos, quedando descrito un problema por los valores que toman dichos atributos. De forma sencilla, el problema descrito en un caso se puede considerar como un vector que contiene los valores que toman dichos atributos:

$$\text{Problema} = (at_1, at_2, \dots, at_n) \quad (22.2)$$

Estos atributos pueden ser cuantitativos (por ejemplo, Peso=650 Kg) o cualitativos (por ejemplo, Color=rojo) dependiendo de la naturaleza del problema. Por ejemplo, supongamos que estamos diseñando un SRBC para una *Agencia Inmobiliaria* cuyo

objetivo será el de indicarnos un coste orientativo del alquiler para una vivienda dependiendo de sus características. Por lo tanto, para obtener el alquiler, un caso debe describir las características de una vivienda y su precio de alquiler. Por ejemplo, podríamos tener casos que indicaran que una vivienda de 2 dormitorios con un aseo y $89 m^2$ en la *Calle Afueras nº 142* que se encuentra en la periferia es una vivienda económica, o el caso de una vivienda de lujo de 5 dormitorios, 3 aseos y $175 m^2$ en la *Plaza Mayor*, ubicada en el centro de la ciudad, puede ser considerada como una vivienda de lujo (realmente lo que hemos definido es el rango de precios en los que podemos valorar dicha vivienda). Esto se podría definir de la siguiente manera:

$$\text{Caso Vivienda} = \text{Problema Vivienda} \times \text{Solución Vivienda}$$

$$\text{Problema Vivienda} = (\text{dormitorios}, \text{aseos}, \text{superficie}, \text{localización})$$

$$\text{Solución Vivienda} = (\text{categoría-precio})$$

$$\text{Cas01 : Calle Afueras nº 142} = ((2, 1, 89, \text{periferia}), (\text{vivienda_económica}))$$

$$\text{Cas04 : Plaza Mayor nº 3} = ((5, 3, 175, \text{céntrica}), (\text{vivienda_lujo}))$$

Sin embargo, en ciertas ocasiones es necesario obtener representaciones más flexibles. Por ejemplo, podríamos permitir que los casos tengan un conjunto diferente de atributos o incluyan conocimiento del contexto (descripción adicional de factores externos), del problema (por ejemplo, una cuantificación de la importancia de cada atributo dentro del problema o relaciones entre atributos), una valoración del caso (es decir, si la solución del caso es un éxito o es un fracaso)... etc. Trabajos más recientes están explorando la posibilidad de definir casos mediante una aproximación basada en objetos, pero en la actualidad la mayor parte de las aplicaciones comerciales siguen la aproximación comentada en esta sección. En definitiva, como podemos apreciar, establecer la estructura y descripción de los casos es una tarea fundamental en el diseño de un SRBC que debe ser analizada con suma atención, puesto que esta decisión repercutirá en el funcionamiento y eficiencia del SRBC.

22.3.2 Librería de casos

La LC es la parte de un SRBC que se encarga del almacenamiento y organización de los casos. La organización de los casos en la LC es crucial para la fase de recuperación de casos similares. Esta fase se puede considerar dividida en dos etapas. Por un lado, tenemos la búsqueda de un subconjunto de casos que puedan ser aplicados al problema planteado. Posteriormente, habrá que aplicar medidas de similitud para seleccionar, de ese subconjunto de casos, aquél más próximo al problema planteado (esta última etapa será tratada en profundidad en la sección 22.3.3). Para que la búsqueda sea eficiente, es necesario diseñar una estructura de almacenamiento adecuada. En el diseño de la estructura de almacenamiento hay que tener en cuenta aspectos claves como: el número de casos a almacenar, requerimientos del dominio de aplicación, etc. Otro objetivo que se debe tener en mente es que la estructura de almacenamiento también debe permitir una inserción eficiente de nuevos casos. Esto es crucial para la incorporación de nuevos casos (aprendizaje), que será analizada en la sección 22.4.4.

Para garantizar una recuperación eficiente de los casos relevantes, se necesitan técnicas de indexación, entendiendo por eficiencia tanto los aspectos relacionados con el tiempo de recuperación como la garantía de que ningún caso relevante quede fuera de la búsqueda. Los índices es una estructura de datos que se mantiene en memoria y que puede ser recorrida de forma rápida para evitar la búsqueda directa en el contenido de la LC. Normalmente, estarán constituidos por combinaciones de los descriptores más importantes de los casos para permitir diferenciar unos casos de otros de manera poco costosa. Los índices hacen referencia a:

- La información indexada de los casos para facilitar su búsqueda.
- La información contextual no contenida en el caso, pero que es relevante para su aplicación.
- Información relativa acerca de cuáles son las características más significativas, es decir, aquellas que permiten determinar la importancia de los atributos dentro del mismo caso.

El establecimiento de índices no es una tarea sencilla, no obstante es posible utilizar técnicas automáticas de indexación. Los procesos automáticos de indexación han demostrado ser altamente predictivos, siempre y cuando exista un conjunto de características muy bien definidas o cuando se trabaje en dominios reducidos. Actualmente, muchas de las herramientas para el desarrollo de SRBC incorporan técnicas automáticas de generación de índices a partir de atributos, normalmente basadas en técnicas de inducción (para un análisis detallado de este tipo de técnicas véase el capítulo 17). Sin embargo, no siempre es posible emplear métodos automáticos, siendo necesarios los procesos manuales de indexación. Por ejemplo, en un dominio tan complejo como el de la medicina, los atributos que componen el problema de un caso (síntomas y hallazgos clínicos que definen la historia de un paciente) están sujetos a matices del conocimiento médico que no es fácil indexar de forma automática. Como ya se ha mencionado, desde el punto de vista conceptual (véase la sección 22.3.1) los casos se estructuran en problema y solución. No obstante, desde el punto de vista del almacenamiento en la LC no siempre conviene conservar esta estructura, siendo más oportuno establecer modelos que mejoren la búsqueda e inserción de nuevos casos. Esencialmente se distinguen dos estructuras de almacenamiento:

- **Memoria Plana.** Esta estructura de almacenamiento presenta los casos de forma secuencial (listas o arrays), donde cada elemento de la secuencia es un caso completo (véase la Figura 22.2 izquierda). La ventaja de este tipo de almacenamiento es que la inserción de casos es sencilla (a menudo de orden constante). Sin embargo, la búsqueda de casos es poco eficiente, por lo que siempre sería necesaria la utilización de técnicas de indexación para disminuir el coste que supondría la búsqueda.
- **Memoria Jerárquica (MJ).** En este caso, la estructura de almacenamiento es un grafo, donde cada nodo almacena ciertos atributos (compartidos o no) por varios casos (véase la Figura 22.2 derecha). Normalmente, se suele utilizar

estructuras de árbol, donde los nodos interiores son atributos comunes y las hojas contienen los atributos diferenciados. Así, el camino desde el nodo raíz a la hoja establece todos los atributos del caso, tanto los comunes como los diferentes, que distinguen un caso de otro. A la hora de establecer la estructura arbórea, se selecciona como nodo raíz aquel atributo más discriminante. La ventaja fundamental de este tipo de estructura de almacenamiento es la mejora en la búsqueda de casos. Sin embargo, la incorporación de nuevos casos resulta más compleja que en una representación secuencial.

Desde el punto de vista de las ciencias que estudian el RBC (computación, psicología cognitiva y filosofía), es el modelo de MJ al que se le está prestando una mayor atención. Así, durante las últimas décadas se han presentado extensiones a dicho modelo. Entre estas extensiones cabe destacar los modelos de **Discriminación de Propiedades**, el de **Memoria Dinámica** y el de **Categoría y Ejemplo**.

El modelo de **Discriminación de Propiedades** mejora el modelo MJ básico en el sentido de que permite la recuperación de casos cuando la descripción del caso a resolver es incompleta. En este modelo, cada nodo de la jerarquía contiene una pregunta acerca de los valores que pueden tomar un atributo, representando los hijos de éstos las respuestas a dichas preguntas. Para que estas jerarquías sean eficientes, los nodos de los primeros niveles hacen referencia a los atributos más discriminatorios, es decir, que discriminen antes los casos que no se buscan.

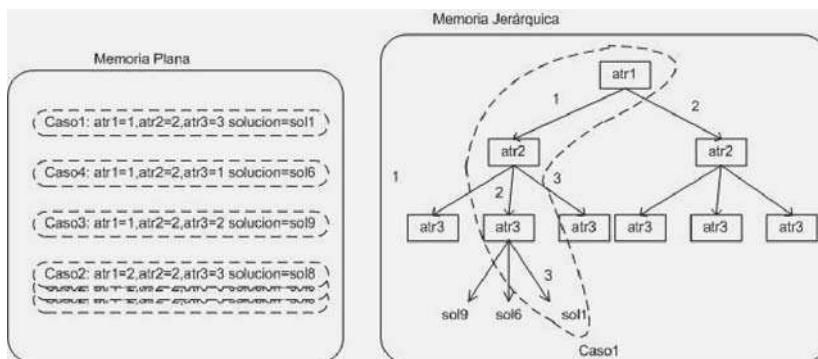


Figura 22.2: Memoria Plana (izq.) y Memoria Jerárquica (der.)

El modelo de **Memoria Dinámica**, es un modelo histórico propuesto por Schank [Schank, 1982] y basado en el concepto de paquetes de organización de modelos o MOPs (del inglés Model Organisation Packages). Uno de los principales problemas del modelo de Discriminación de Propiedades es que el orden de los nodos internos de la jerarquía afecta en la búsqueda del caso. Así, el modelo de Memoria Dinámica organiza los casos en varios grafos (redundante) donde cada uno contiene un orden distinto de los nodos internos (preguntas). Este modelo propone agrupar los casos en

unas nuevas estructuras cuando comparten atributos similares denominadas Episodios Generalizados (EGs). Los EGs están compuestos por normas, índices y casos (véase la Figura 22.3). Una norma es el conjunto de todos los atributos comunes de los casos del EG. Un índice establece los atributos discriminatorios y apuntan a un caso o a otro EG más especializado que agrupa otros casos con atributos comunes más específicos. Buscar un caso consiste en ir recorriendo las estructuras y responder a las preguntas a partir de los datos disponibles.

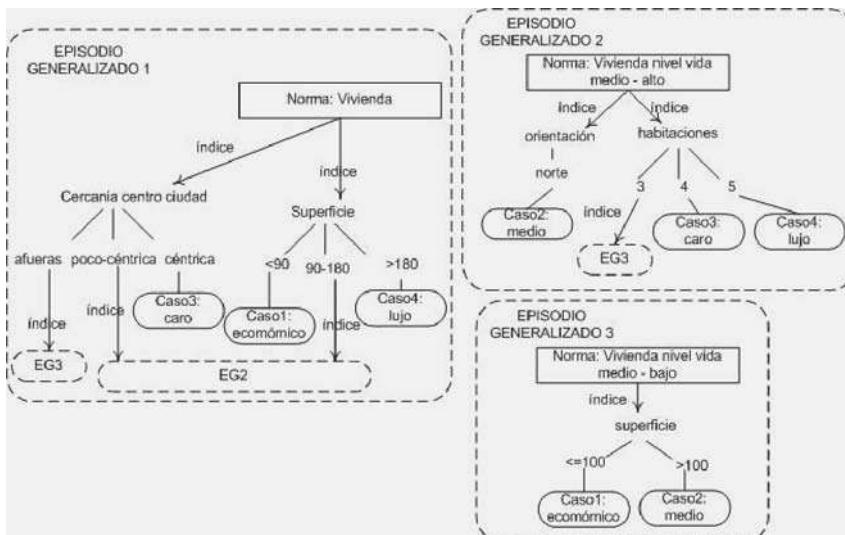


Figura 22.3: Episodio Generalizado.

En el modelo de **Categoría y Ejemplo**, propuesto por Bareiss y Porter [Bareiss, 1988], los casos están distribuidos en una red que representa al mismo tiempo las descripciones de cada caso y el conocimiento del dominio. En este modelo existen tres tipos de elementos: categorías, casos y enlaces. Las categorías agrupan casos similares y los enlaces relacionan los casos con: sus propios atributos, otros casos similares, otros casos no similares debido a una característica discriminatoria. Buscar un caso en este modelo consiste en ver qué casos comparten más atributos comunes a partir de los datos disponibles.

22.3.3 Determinación de casos similares

En la sección anterior, hemos analizado técnicas que permiten la búsqueda eficiente de un conjunto de casos que pueda ser aplicado al problema que se intenta resolver. Sin embargo, hay que tener en cuenta que los modelos analizados sólo permiten hacer búsquedas exactas de atributos. Una vez que tenemos ese conjunto de casos, hay que

determinar qué caso, o casos, son los más similares y que, por lo tanto, se pueden utilizar para resolver el problema planteado. Para llevar a cabo esta fase, resulta imprescindible establecer la manera de medir computacionalmente la similitud entre casos. Es bien sabido en el campo de la psicología que la determinación de un grado de similitud es un proceso complejo que no puede ser estudiado de forma aislada y que depende en cierta medida de factores externos. Además, la propia similitud puede ser interpretada de diferentes maneras: el grado de relación de las características, la situación contextual o la dificultad que supondría su fusión (adaptación). En este capítulo asumiremos la interpretación clásica, es decir, la medida de similitud es el grado de cercanía existente entre los atributos que componen el caso, y nos permitirá cuantificar sus correspondencias parciales. Por lo tanto, las medidas de similitud son las herramientas de las que dispone un SRBC para cuantificar cuán similares son los casos. Así, una medida de similitud contiene conocimiento, más o menos sofisticado, sobre la forma de cuantificar dicha similitud (tradicionalmente entre pares de casos).

Una forma de definir la similitud entre casos consiste en la determinación de la similitud de los atributos o características del caso. Para ello, hay que tener en cuenta que no todos los atributos que componen los casos se pueden comparar de la misma forma. Por ejemplo, en el problema de la *Agencia Inmobiliaria* (véase la sección 22.3.1) no se puede comparar de la misma manera el atributo *dormitorios*, que es un valor cuantitativo (numérico), que el atributo *localización*, que es un valor cualitativo (nominal). Ni tampoco es razonable utilizar siempre el mismo método para establecer la similitud entre los valores del atributo *aseos* y del atributo *superficie*, ya que a pesar de ser ambos valores cuantitativos (valores enteros positivos), se encuentran en unidades y rangos diferentes (la superficie suele encontrarse entre 50 y 300 metros cuadrados, mientras que los aseos suelen ser entre 1 y 4). Una forma de solventar estos problemas consiste en considerar dos medidas de similitud: **similitud local** y **similitud global**. La **similitud local** hace referencia a la medida que cuantifica la similitud entre atributos del mismo tipo, mientras que la **similitud global** mide la similitud para el conjunto de todos los atributos o características de los casos. Normalmente, la similitud global se define en función de las similitudes locales.

Para definir las funciones de similitud, vamos a suponer que queremos determinar el grado de similitud entre dos casos, c_1 y c_2 , donde todos sus atributos son numéricos y que se representan de la siguiente forma:

$$c_1 = (at_1^1, at_2^1, \dots, at_n^1) \quad (22.3)$$

$$c_2 = (at_1^2, at_2^2, \dots, at_n^2) \quad (22.4)$$

Para calcular la similitud entre casos (similitud global) se suelen utilizar técnicas basadas en k-vecinos o kNN (véase el capítulo 16), para lo que es necesario definir una función de distancia entre los valores de cada atributo. Esto es posible ya que existe una relación inversamente proporcional entre el concepto de distancia y la similitud. Es decir, si calculamos la similitud entre dos casos, cuanta menor sea la distancia existente entre los valores sus atributos, mayor es la similitud entre estos. Para ello, se calcula la distancia entre los casos combinando las distancias sobre los atributos.

Una forma sencilla para calcular la similitud (global) entre casos cuyos atributos son cuantitativos (numéricos) es a través de la distancia euclídea entre sus atributos. Para ello se suelen establecer unos pesos, w_k , que determinan la importancia de cada tipo de atributo dentro del caso. Por ejemplo, en el problema de la *Agencia Inmobiliaria*, a la hora de determinar la similitud entre dos viviendas, se podría considerar un factor más determinante el hecho de que tengan el mismo número de dormitorios en vez de que tengan el mismo número de aseos. De esta forma, la similitud local se puede definir como la distancia entre los casos c_i y c_j :

$$D(c_i, c_j) = \frac{\sqrt{\sum_{k=1}^n w_k d_k(at_k^i, at_k^j)^2}}{\sum_{k=1}^n w_k} \quad (22.5)$$

donde d_k la función de distancia euclídea entre los valores de los atributos de tipo k y w_k son los pesos que establecen la importancia de cada atributo dentro del caso. En este caso, la similitud global se puede definir como:

$$S(C_i, C_j) = \frac{1}{D(C_i, C_j)} \quad (22.6)$$

Por ejemplo, supongamos que en el problema de la *Agencia Inmobiliaria* tenemos en la LC, tres casos:

Caso1=((2,3,150,'céntrico'),('caro'))
Caso2=((5,1,160,'afuera'),('económico'))
Caso3=((2,1,85,'céntrico'),('asequible'))

Figura 22.4: Librería de casos.

Y tenemos como caso de entrada una descripción parcial del apartamento deseado, estableciendo sólo algunos atributos del caso:

$$C_{in} = ((4, 3, -, -), (\emptyset)) \quad (22.7)$$

donde “-” representa un valor no determinado de un atributo, y el símbolo \emptyset de la solución indica que es un caso no resuelto. Además, los diferentes atributos del apartamento (*dormitorios*, *aseos*, *superficie* y *localización*) se han clasificado según su importancia, estableciendo los siguientes pesos:

$$w_{dormitorios} = 0,3 \quad (22.8)$$

$$w_{aseos} = 0,05 \quad (22.9)$$

$$w_{superficie} = 0,4 \quad (22.10)$$

$$w_{localización} = 0,25 \quad (22.11)$$

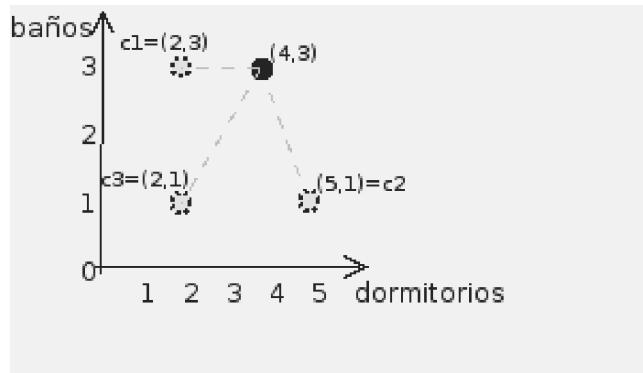


Figura 22.5: Distancia entre atributos.

El resultado de las distancias entre C_{in} y los casos almacenados ($D(C_{in}, C_i)$) calculadas según las expresiones 22.5 y 22.6 pueden verse en la Tabla 22.1. Como podemos observar, el Caso 2 sería el elegido de entre todos los casos como el más similar.

	Caso 1	Caso 2	Caso 3
dist. dormitorio	2	1	2
dist. aseos	0	2	2
Distancia total (D)	1,7142857	1,142857	2
Similitud Global (S)	0,583333	0,875	0,5

Tabla 22.1: Cálculo de las similitudes locales ($d_{dormitorios}$ y d_{aseos}), distancia total D y similitud global S para los diferentes casos.

En la Figura 22.5 podemos apreciar que, aunque la distancia entre atributos indica la similitud entre los casos (similitud local), los pesos pueden determinar lo contrario respecto a la similitud global. Como podemos observar, una de las mayores dificultades a la hora de describir dicha función radica en la valoración de la importancia de cada atributo (pesos), siendo estos dependientes del dominio de aplicación concreto. Examinando el ejemplo desarrollado sobre la similitud en el problema de la *Agencia Inmobiliaria*, a la medida de similitud propuesta anteriormente se le pueden poner dos objeciones. La primera es que no considera aquellos atributos que no están presentes en el caso (por ejemplo el atributo *superficie*). La segunda es que está restringido únicamente a atributos numéricos (por ejemplo, el atributo *situación* no contribuye a la medida de similitud). Para solventar estos problemas, necesitamos una medida de similitud global mucho más flexible, como la distancia euclídea heterogénea. La distancia euclídea heterogénea básica se define como:

$$d_{at}(at_i, at_j) = \begin{cases} 1 & \text{si } at_i \vee at_j \text{ faltan} \\ 0 & \text{si } at_i \wedge at_j \text{ nominal } \wedge at_i = at_j \\ 1 & \text{si } at_i \wedge at_j \text{ nominal } \wedge at_i \neq at_j \\ \frac{|at_1 - at_2|}{rango_{at}} & \text{si } at_i \wedge at_j \text{ lineal} \end{cases} \quad (22.12)$$

Esta distancia local, calcula un valor de distancia normalizado. Es decir, la distancia mínima es cero y la máxima posible es 1. Así, dependiendo de si el atributo es nominal (cadena de caracteres) o numérico, calcula la similitud de forma diversa. Además, penaliza aquellos atributos ausentes dándoles el valor máximo.

22.3.3.1 Tipos de distancias

La medida de similitud descrita anteriormente es sólo un ejemplo de medida de similitud numérica a partir de pesos de atributos y basado en el concepto de distancia euclídea. A la hora de definir funciones que calculen la distancia entre elementos, en el capítulo 16 se hace una clara distinción entre los conceptos de índice de proximidad, distancia e índice de semejanza, según las propiedades matemáticas que presentan (reflexividad, simetría, etc.). Sin embargo, en el dominio RBC se denomina distancia de forma indistinta a cualquiera de ellas. No obstante, existen otras muchas formas de clasificar medidas dependiendo del dominio concreto y la naturaleza de los atributos.

Entre las funciones de distancia que asumen *atributos cuantitativos*, la función más sencilla es la de **Manhattan** que se define como:

$$D_{Man}(C_i, C_j) = \sum_k w_k d_k(at_k^i, at_k^j) \quad (22.13)$$

Se suele utilizar cuando se quiere simplificar la complejidad de cálculo. Una función un poco más compleja, y más utilizada, es la distancia **Euclídea Ponderada**:

$$D_{Euc}(C_i, C_j) = \sqrt{\frac{\sum_k (w_k d_k(at_k^i, at_k^j))^2}{\sum_k w_k}} \quad (22.14)$$

Todas estas medidas se derivan de la métrica de **Minkowski** que, en términos de RBC, se describe así:

$$D_{Minkowsky}^M(C_i, C_j) = \sqrt{\frac{\left(\sum_k (w_k d_k(at_k^i, at_k^j))^M \right)^{\frac{1}{M}}}{\sum_k w_k}} \quad (22.15)$$

Así, con $M=1$ y $M=2$ se caracterizan las distancias de Manhattan y la Euclídea. Otras métricas de similitud numéricas que pueden encontrarse son las basadas en

operadores estadísticos de la teoría clásica de la probabilidad. De este tipo, una de las más utilizadas en RBC es la basada en el estadístico χ^2 , que calcula cuán distinta es una población p_x respecto a p_y , y se calcula de la siguiente forma:

$$d(p_x, p_y) = \sum_{k=1}^m \left(\frac{x_k - m_k}{m_k} \right)^2 \quad (22.16)$$

$$\text{con } m_k = \frac{at_k^x + at_k^y}{2} \quad (22.17)$$

Medida	Tipo	Fórmula	Utilización
Manhattan $D_{Man}(C_i, C_j)$	Numérico	$\sum_k w_k d_k(at_k^i, at_k^j)$	Poco cálculo requerido
Euclides $D_{Euc}(C_i, C_j)$	Numérico	$\sqrt{\sum_k \left(w_k d_k(at_k^i, at_k^j) \right)^2}$	Universal
Minkowski $D_{Min}^M(C_i, C_j)$	Numérico	$\left(\sum_k \left(w_k d_k(at_k^i, at_k^j) \right)^M \right)^{\frac{1}{M}}$	Universal
Chi cuadrado $d(p_x, p_y)$	Numérico	$\sum_{k=1}^m \left(\frac{x_k - m_k}{m_k} \right)^2$	Gran conjunto de atributos
Clark $D_{cla}(c_x, c_y)$	Numérico	$\sum_{k=1}^m \frac{ x_k - y_k ^2}{ y_k - x_k ^2}$	Sin pesos
Canberra $D_{cam}(c_x, c_y)$	Numérico	$\sum_{k=1}^m \frac{ x_k - y_k ^2}{ y_k + x_k ^2}$	Sin pesos
Hamming	General	Calcula el número de atributos que son idénticos	Análisis de diferencias
Disimilitud (o Tversky)	General	contabiliza la parte común y la diferente $\frac{\alpha(\text{común})}{\alpha(\text{común}) + \beta(\text{diferente})}$ donde α, β son pesos	Cuantificar la NO similitud
Weber	General	Analiza el hecho de que ciertos atributos se encuentren y otros no $\alpha f(at_{c_x} \cap at_{c_y}) - \beta f(at_{c_x} - at_{c_y}) - \omega f(at_{c_y} - at_{c_x})$	Análisis de diferencias
Edición (o Levenshtein)	Nominal	Calcula el coste de transformar caso en otro mediante la combinación de operadores de edición de caracteres (añadir, eliminar, mover)	Los casos son secuencias de caracteres
Superficiales	Nominal	Procesamiento de secuencias de caracteres	Análisis de texto como conjunto de caracteres
Estructurales	Nominal	Analiza topologías y estructuras	El caso es texto estructurado
Semánticas	Nominal	Distancia entre términos mediante tesauros, diccionarios y ontologías. Depende del dominio concreto.	Conocimiento del dominio

Tabla 22.2: Listado de las medidas de similitud/distancias más utilizadas en RBC.

Otra medida que se puede utilizar es la **distorción de Hamming** que simplemente calcula cuántos atributos tienen el mismo valor y cuántos no. Para el caso en el que tenemos atributos cuantitativos, existen diversas alternativas:

- **Similitudes superficiales**, que se basan en la correspondencia sintáctica del texto. El procesamiento más sencillo es hacer una comparación exacta (del inglés *exact matching*) y establecer si son o no iguales las cadenas. Otra posibilidad es utilizar la **distorción de edición** (o de *Levenshtein*), que calcula el esfuerzo que supone transformar una atributo cuantitativo en otro mediante la utilización de tres operadores básicos: inserta nuevo carácter, elimina carácter o desplaza el carácter. Los algoritmos para la distancia de edición calculan el coste del conjunto mínimo de operadores para obtener, dada una cadena de caracteres, otra.
- **Similitudes estructurales** se basan en métricas para comparar estructuras y topología de los datos. En esta categoría se incluyen todas aquellas técnicas basadas en modelos y los algoritmos de comparativa de grafos.
- **Similitudes semánticas** son aquellas en las que los elementos nominales tienen un significado dependiente del dominio. Para medir la semántica se utilizan diccionarios, tesauros y ontologías con el fin de cuantificar las relaciones entre los elementos.

Además de las aquí nombradas, existe un amplio abanico de medidas dependiendo de la naturaleza concreta de los datos (por ejemplo, si se considera el factor tiempo) o de algún objetivo particular (por ejemplo, calcular cuánto no se parece). Una selección de las principales estrategias para calcular similitud en sistemas RBC se puede ver en la Tabla 22.2.

22.4 El ciclo RBC. Etapas-RE

Una de las principales características del RBC es que, frente a otras soluciones que ofrece la IA, además de resolver un problema, proporciona una descripción ordenada de las etapas que deben realizarse para la obtención de la solución. En otras palabras, nos indican cómo resolver un problema determinado, buscando en LC problemas similares, adaptando sus soluciones al problema y finalmente retroalimentando la LC con el nuevo problema resuelto. Estas etapas están descritas en un proceso cíclico denominado ciclo RBC [Aamodt y Plaza, 1994] que cubre las cuatro etapas principales del RBC (véase la Figura 22.6), denominadas *etapas-RE*:

1. **Recuperar** casos similares a la descripción del problema a resolver.
2. **Reutilizar** (adaptar) la solución sugerida por alguno de los casos similares, adaptándola para que resuelva el problema.
3. **Revisar** que la nueva solución sea correcta.

4. **Retener** (almacenar) partes del problema y la nueva solución como un nuevo caso en la LC.

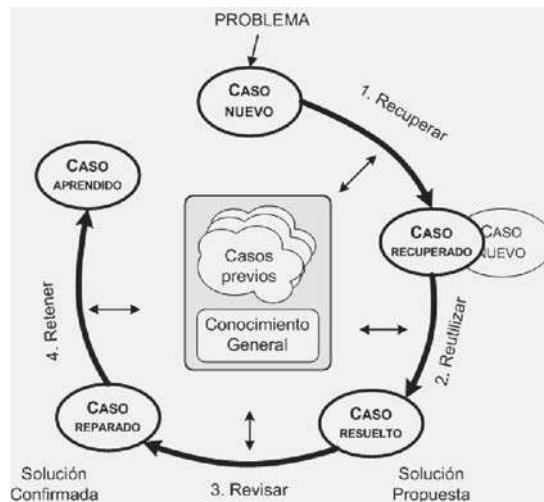


Figura 22.6: El ciclo RBC propuesto en [Aamodt y Plaza, 1994].

Una de las particularidades del ciclo RBC es que se describe qué hay que hacer pero no cómo hacerlo. Por ello, existe cierta controversia entre los investigadores, puesto que algunos consideran el RBC como una metodología para resolver problemas y no como un método de resolución de problemas propiamente dicho. Sin embargo, algunos autores han analizado desde la perspectiva del modelado de conocimiento el ciclo RBC. En [Aamodt y Plaza, 1994] se puede ver una descomposición de tareas y métodos en los que se puede descomponer el ciclo RBC (véase la Figura 22.7). Aparte de este ciclo, existe una tarea denominada **mantenimiento** que, si bien no es parte integrante del proceso RBC, es de vital importancia en el ciclo de vida del SRBC. En los siguientes subapartados se describe cada una de las tareas en detalle.

22.4.1 Recuperar

El objetivo de la etapa de *Recuperación* es rescatar de la LC aquellos casos considerados suficientemente similares al problema que se desea resolver, que constituye la entrada de esta etapa. Un paso previo al proceso de búsqueda, y una vez que tenemos descrito el problema a resolver, consiste en la identificación de las características relevantes del problema planteado, es decir, seleccionar los atributos relevantes mediante algún tipo de filtro y eliminar elementos poco descriptivos que pudiesen llevar a búsquedas frustradas. Algunos sistemas también consideran un proceso de abstracción como paso previo al proceso de búsqueda.

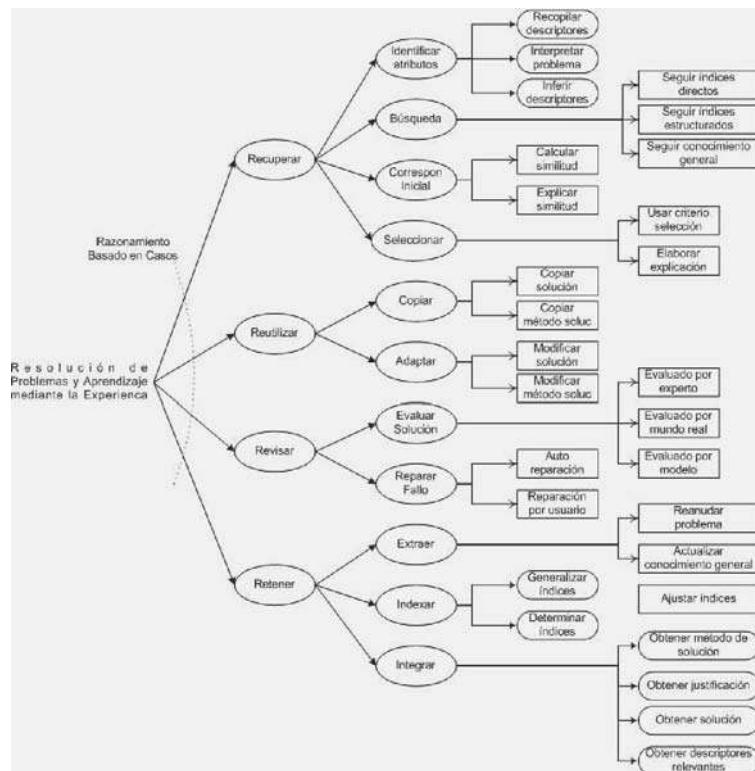


Figura 22.7: Las tareas y métodos del RBC propuesto por [Aamodt y Plaza, 1994].

Tradicionalmente la búsqueda de casos más similares se subdivide en dos pasos. En primer lugar, se hace una búsqueda de aquellos casos similares que sean plausibles (denominado correspondencia inicial). Para obtener estos casos de forma correcta y eficiente, el modo de almacenamiento en la LC (memoria plana, jerárquica...) y las técnicas de indexación juegan un papel fundamental (véase la sección 22.3.2). Una vez obtenido un conjunto de casos candidatos, el segundo paso (denominado selección) consiste en seleccionar de entre los casos candidatos aquellos que sean más similares a la descripción del problema. En esta etapa es necesario establecer cuáles son las medidas de similitud y distancias más adecuadas en el dominio concreto de aplicación (véase la sección 22.3.3). Una de las técnicas más utilizadas para seleccionar los casos más similares es mediante la estrategia de los *k vecinos más cercanos* (kNN), utilizando como función de evaluación la función de similitud. Es decir, seleccionando de entre todos los casos candidatos, los *k* más similares al problema que deseo resolver. Esta técnica tiene la ventaja de ser muy simple, sin embargo debe calcular la similitud del problema a resolver con cada uno de los casos candidatos.

22.4.2 Reutilizar

Una vez que se obtienen de la LC los casos más similares al problema planteado, el sistema intentará utilizarlos para dar una solución. En algunos casos, bastará simplemente con proporcionar alguna de las soluciones que establecen los casos recuperados. Sin embargo, existen situaciones en las que las soluciones propuestas en los casos recuperados no son aplicables directamente. En estos casos, el sistema deberá *adaptar* las soluciones almacenadas en los casos recuperados al problema planteado, determinando cuáles son las diferencias y aplicando algún proceso que tenga en cuenta dichas diferencias y proporcione una solución concreta. Por lo tanto, esta adaptación puede entenderse como la reutilización directa de una solución anterior con alguna modificación o como la reutilización de un método ya empleado para resolver un problema anterior. Podemos diferenciar dos aproximaciones a la hora de realizar la adaptación:

- En la **Adaptación transformacional**, se intenta reutilizar la solución del caso recuperado para obtener una nueva solución mediante un conjunto de operadores que transforman la solución (véase la Figura 22.7). En esta aproximación no se analiza cómo se resolvió el problema, sino que se busca una equivalencia para poder resolver el problema.
- La **Adaptación derivacional** obtiene el método por el cual se obtuvo la solución, a partir de los casos recuperados, y lo aplica para resolver el problema y obtener la nueva solución (véase la Figura 22.7). Evidentemente, en este caso se requiere que la secuencia de pasos que da lugar a la solución se almacene como un componente más del caso.

Existen diferentes métodos para reutilizar los casos recuperados (véase la Figura 22.7); a continuación describiremos tres de los métodos más utilizados:

- La **reinstanciación** es la forma más sencilla de adaptación, puesto que consiste simplemente en copiar la solución del caso recuperado como nueva solución del problema. Se utiliza principalmente cuando el grado de similitud es muy alto entre el caso recuperado y el problema. Su dificultad reside en establecer los umbrales de aceptación del grado de similitud. En el ejemplo de la *Agencia Inmobiliaria* (véase la Figura 22.4), el caso más similar era el Caso 2 cuya solución era que el precio del apartamento se encontraba en la categoría *económico*. Una adaptación por reinstanciación asignaría esta misma categoría como solución del nuevo caso.
- La **sustitución** de valores consiste en reemplazar valores de partes de la solución de los casos recuperados para obtener la solución del problema. Aquellas partes que contradigan los requerimientos del problema se sustituyen, por lo que se requiere de otras técnicas de modelado de conocimiento como la utilización de satisfacción de restricciones (véase el capítulo 10), reglas, o el ajuste de parámetros. El método de sustitución se suele emplear cuando estamos utilizando una LC sólida. En el ejemplo de la *Agencia Inmobiliaria* la solución está formada por un único elemento. Si supusiésemos que la solución está formada

por un conjunto de atributos ($solucion = (s1, s2, s3)$), este tipo de adaptación intentaría sustituir en el nuevo caso alguno de los atributos solución encontrados en el caso más similar (Caso 2), sin contradecir las restricciones del dominio.

- El método de **reparación dirigida por modelos** se aplica cuando en los casos recuperados no se encuentra ninguna solución que satisfaga las restricciones del problema planteado. Este método obtiene una nueva solución a partir de las restricciones y características de la solución que se espera obtener. Para ello intenta detectar qué componentes de la solución del caso recuperado se pueden reutilizar, reparando el resto mediante una transformación. Esta transformación es dirigida mediante el conocimiento representado en modelos o heurísticas. En el ejemplo de la *Agencia Inmobiliaria* no sería conveniente utilizar este tipo de adaptación, puesto que no tenemos conocimiento adicional que establezca restricciones del dominio. Sin embargo, supongamos que existe dicho conocimiento almacenado en algún tipo de Base de Conocimiento. El caso más similar (Caso 2) aportaría conocimiento nuevo, por ejemplo que la localización del apartamento está en las *afueras*, pudiendo incluirlo en el nuevo caso para obtener una solución consistente con la Base de Conocimiento.

En general, la etapa de *Reutilización (Adaptación)* es dependiente de otras técnicas intensivas en conocimiento para poder obtener buenos resultados. Existen métodos y técnicas relativamente sencillas, como la reinstanciación o la configuración de parámetros, sin embargo en otras muchas ocasiones es necesaria la utilización de estrategias y modelos de mayor coste y que aumentan la complejidad de los sistemas RBC. Algunos expertos consideran que muchos SRBC sin etapa de *Reutilización* (denominados Sistemas de Recuperación de Casos) obtienen muy buenos resultados y que debido a la complejidad de esta etapa es conveniente no tenerla siempre en cuenta. Sin embargo, otros autores mantienen que sin esta etapa, los sistemas RBC se convierten en un simple proceso de búsqueda avanzada.

22.4.3 Revisar

El objetivo consiste en decidir si la solución propuesta en la etapa anterior es aceptable como solución del problema planteado. Si tras la evaluación de la solución ésta es aceptada, se generará un nuevo caso a partir del par problema-solución, que pasará a la siguiente etapa, *Retener*. Si la solución no es aceptable, se intentará reparar la solución realizando los cambios pertinentes. Esta etapa es necesaria ya que la etapa anterior, *Reutilizar*, puede obtener soluciones incorrectas. De esta forma, se garantiza que ningún caso incorrecto pueda ser incluido en la LC, posibilitando de esta forma un proceso de aprendizaje correcto. En la mayoría de las ocasiones esta etapa está supervisada por un experto en el dominio concreto, bien realizando esta tarea de forma manual o semiautomática.

22.4.4 Retener

En esta etapa el objetivo consiste en, una vez que se tiene un caso resuelto y revisado, incorporarlo a la LC, terminando de esta manera la fase de aprendizaje. Antes de introducirlo en la LC, debemos decidir qué información, de toda la que proporciona el caso resuelto, se va a incluir. En primer lugar, resulta evidente que este nuevo caso tiene que incluir los descriptores del problema (los atributos) y de la solución. Sin embargo, algunos de estos descriptores pueden ser poco o nada relevantes para el conjunto global del caso, por lo que habitualmente se hace una selección de los mismos, proceso denominado *extracción*. En segundo lugar, hay que considerar la información obtenida en la etapa de *Revisión*, por lo que es muy útil incorporar en el caso la metainformación relativa a una posible reparación de la solución (a modo de recordatorio) para aplicarla en un futuro. Algunos sistemas también contemplan la posibilidad de fusionar el nuevo caso con alguno ya existente en la LC, dando lugar a un único caso generalizado.

Finalmente, también hay que considerar que un caso está almacenado en la LC para ser recuperado en el futuro. Para ello hay que, en primer lugar analizar qué atributos pueden formar parte de la indexación del caso, y actualizar el índice de acuerdo con dichos atributos, es decir, se tiene que *indexar* el nuevo caso. En segundo lugar, hay que almacenar el caso físicamente en el propio almacenamiento estructurado en la LC, es decir, tenemos que *integrar* el nuevo caso en la estructura de almacenamiento. Evidentemente, la eficiencia de esta fase depende de la complejidad tanto del índice como de la estructura de almacenamiento.

22.4.5 Mantenimiento

Como hemos podido ver, los sistemas RBC suelen incorporar casos adaptados que han resuelto un nuevo problema, han superado un proceso de supervisión (etapa *Revisar*) y son almacenados (etapa **Retener**) e indexados. Sin embargo, la incorporación de estos casos, a pesar de ser correctos, pueden afectar negativamente la eficiencia global del SRBC, por lo que es conveniente realizar un mantenimiento correcto de la LC. El primer problema que hay que abordar es el de la redundancia de casos. A pesar de que el duplicado de casos no implica la obtención de soluciones incorrectas, la redundancia resulta innecesaria, ralentiza la búsqueda de casos y aumenta la recuperación de casos sin aportar soluciones alternativas. Para evitar esta situación resulta aconsejable establecer medidas para evitar redundancias de forma automática, como por ejemplo las comprobaciones de casos de la LC, o manuales, como por ejemplo consultas a expertos.

Una segunda medida en el mantenimiento de los sistemas RBC consiste en realizar un análisis estadístico relativo a la recuperación de casos. De esta forma, se podría descubrir que un caso es recuperado un número excesivo de veces, o por el contrario ninguna vez. En la primera situación el caso es probablemente muy poco específico y satisface la mayoría de las búsquedas. En la segunda situación, el caso no ha sido recuperado nunca, lo que indica que no es relevante para el dominio a pesar de ser concreto. Otros estudios estadísticos aconsejables se centran en el estudio del tiempo

medio de recuperación de casos o las veces que fue posible realizar la adaptación de casos de forma exitosa.

22.5 Aplicaciones de los sistemas RBC

Los sistemas de RBC se han podido aplicar con éxito a gran cantidad de dominios, desarrollando gran cantidad de tareas, tanto en el campo académico como en el de la industria. Como ya hemos mencionado a lo largo de este capítulo, los SRBC son idóneos para desarrollar tareas analíticas, concretamente clasificación y diagnósticos. Sin embargo, su aplicación para tareas de síntesis, como por ejemplo el diseño por configuración y la planificación, aunque posible, requiere de un mayor esfuerzo. A medida que las tareas y los dominios para los que se desarrollan SRBC se hacen más complejos, se hace necesario recurrir a otros paradigmas de la I.A. (sistema de reglas, modelos,...) para la representación de conocimiento, recuperación o adaptación de casos, conformando los denominados *sistemas híbridos*. Dentro del ámbito de las académicas académicas podemos destacar:

- **SRBC para el Diagnóstico.** El diagnóstico es la tarea en las que se han centrado la mayoría de los desarrollos de SRBC (un análisis más detallado de la tarea de diagnóstico se puede ver en el capítulo 13). En algunas ocasiones los sistemas RBC se limitan a intentar clasificar las evidencias/fallos que describen los casos, en otras ocasiones existe un proceso de inferencia con el fin de obtener una hipótesis diagnóstica. En particular, algunos dominios como el médico son idóneos para los sistemas RBC puesto que los médicos suelen recordar casos clínicos de pacientes para apoyar sus diagnósticos y la complejidad del dominio clínico hace que, en muchas ocasiones, no se pueda establecer un modelo. Dos ejemplos históricos de sistemas RBC para el diagnóstico son:
 - **PROTOS** [Bareiss, 1988]. Sistema centrado en el diagnóstico de las afecciones del oído. Este sistema intenta clasificar en una de las categorías diagnósticas las evidencias de un paciente encontradas en resultado de pruebas, la historia clínica y los síntomas.
 - **CASEY** [Koton, 1989]. Sistema inteligente centrado en el dominio de los fallos cardíacos que combina *Razonamiento Basado en Casos y Sistemas Basados en Reglas*. Este sistema recupera los casos más similares y determina cuáles son las diferencias entre los casos recuperados con el caso de entrada. Si aparecen pocas diferencias, utiliza un conjunto de operadores de adaptación para obtener un diagnóstico. Si las diferencias son importantes y no es posible la adaptación de la solución, CASEY aplica el módulo basado en reglas para obtener un posible diagnóstico.
- **SRBC para la Planificación.** La planificación se puede definir como el proceso que nos permite obtener el conjunto de pasos que permiten alcanzar un objetivo previamente establecido. Normalmente, cuanto más complejo es el dominio más largos resultan los planes y, por lo tanto, más complejo es realizar la tarea de

planificación (para un análisis más detallado de la tarea de planificación véase el capítulo 13). Los sistemas RBC son útiles para estas tareas puesto que en vez de intentar obtener un nuevo plan desde el principio, se pueden utilizar planes ya realizados (computacionalmente costosos) e intentar adaptarlo. Ejemplos de este tipo son:

- **CHEF** [Hammond, 1986], es un SRBC cuyo objetivo es la definición de recetas de cocina, entendidas éstas como planes. En CHEF un caso está formado por el conjunto de submetas, los elementos de la receta y la planificación para obtenerlas. Así, en el ciclo RBC se busca en la LC aquel plan que permita obtener submetas. Para adaptar el plan, en primer lugar se reinstancia la planificación antigua pero sustituyendo los nuevos elementos de la receta, y en segundo lugar se realizan labores de ajuste (añadir o eliminar elementos del plan). Finalmente, existe un proceso de reparación por si ha habido algún problema con el fin de obtener un plan consistente.
 - **BOLERO** [López y Plaza, 1993] se diseñó para planificar un tratamiento a partir de la información extraída de la historia clínica. Combina un sistema basado en reglas para diagnosticar la enfermedad que presenta el paciente y una componente de RBC para planificar el tratamiento correspondiente.
- **SRBCs para el diseño.** El diseño implica la construcción de algo nuevo o ya conocido a partir de componentes y dadas unas restricciones para su construcción. Evidentemente, fuera de esta definición se encuentra el proceso de diseño creativo, proceso que hoy por hoy no admite un modelo computacional. Sin embargo, aunque eliminemos al componente creativa, sigue siendo una de las tareas más difíciles a las que se enfrenta la IA. A pesar de la dificultad, existen numerosos ejemplos de sistemas que, aplicando RBC, han conseguido desarrollar una actividad que se puede considerar como diseño. Algunas de las dificultades comunes a los SRBCs en diseño son: la complejidad de los casos, el conocimiento relativo al diseño y la ausencia de formalismos. En estos sistemas los casos suelen implicar un gran conjunto de atributos que no llegan a describir con precisión el caso, o requieren un modelo complejo, lo que complica la etapa de búsqueda y recuperación de éstos. Además, a menudo es necesario adaptar las soluciones, es decir rediseñar. Para ello, es necesario modelar las tareas de diseño que, a menudo, implican utilizar conjuntamente reglas, heurísticas y restricciones que no son siempre fáciles de obtener y aún menos formalizar. Ejemplos históricos de los SRBCs para diseñar son:

- **JULIA** [Hinrichs. y otros, 1991], cuyo objetivo es el diseño de menús para restaurantes. Para ello, obtiene el menú más similar y establece el conjunto de restricciones que el nuevo plato debe cumplir. Para obtener la solución utiliza una librería de métodos de adaptación con el fin de conseguir platos que satisfagan las restricciones.
- **KRITIK** [Goel, 1989], que se construyó como un sistema híbrido para el diseño de pequeños dispositivos electrónicos y mecánicos, combinando

RBC y Razonamiento Basado en Modelos. La parte RBC se utiliza para proponer una solución a partir de diseños ya almacenados, y posteriormente se utilizan los modelos (conocimiento causal) para comprobar si dicho diseño es válido, o bien si es necesaria la adaptación, en cuyo caso sugiere dónde se debe hacer.

- **CADET** [Sycara, 1992] fue diseñado para servir como un asistente en el diseño mecánico. El sistema recupera casos pasados que representan diseños exitosos, al mismo tiempo que evita los diseños que condujeron a algún fallo, tales como aquellos que presentaron problemas con la calidad de los materiales o elevaron los costes de producción. CADET transforma una descripción abstracta del comportamiento que se desea en el dispositivo a diseñar, en una descripción que pueda ser utilizada en la recuperación de diseños previos relevantes para el problema actual y, posteriormente, genera un conjunto de diseños alternativos para un conjunto de especificaciones de diseño.

En el ámbito industrial, la aplicación del RBC también ha dado lugar a sistemas que han cosechado importantes éxitos. Entre éstos, cabe desataracar los siguientes:

- **Clavier.** Desarrollado por la empresa Lockheed en 1897, fue uno de los primeros sistemas RBC comerciales, que estaba orientado al diseño en autoclaves. En la industria aeronáutica se necesita fabricar componentes formados por materiales compuestos que necesitan tratarse simultáneamente a altas temperaturas. Para ello se utiliza una autoclave, un dispositivo similar a una olla a presión de grandes dimensiones que permite someter a altas temperaturas lo que se encuentre en su interior (para realizar esterilización o tratamientos térmicos). Sin embargo, no existen modelos físicos que permitan establecer un proceso exacto para que el tratamiento térmico de estos componentes se realice correctamente y hay que emplear la experiencia de los ingenieros y operadores para realizar esta tarea. Así, Lockheed decidió desarrollar Clavier para asistir a los operadores del autoclave con un SRBC que ayuda en el diseño de este proceso.
- **FormTool.** Desarrollado por la empresa General Electric Plastics en 1994, su objetivo era determinar cuáles son los colorantes que deben ser utilizados para fabricar un plástico de un color determinado. El problema que resuelve este sistema es encontrar la fórmula que componga un nuevo color para un plástico. Para ello, sabiendo ciertas propiedades del nuevo color, se realiza una búsqueda en un catálogo de muestras (y sus composiciones) de colores ya utilizados para encontrar el más parecido. Una vez encontrado, se alterará la fórmula de su composición para obtener el nuevo color. Para realizar este proceso, esta compañía ha utilizado funciones de similitud para seleccionar el color más conveniente, atendiendo a criterios tales como similitud de color, coste de colorantes, densidad óptica del color... etc. Cada una de estas funciones se combina utilizando teoría de conjuntos borrosos y aplicando el algoritmo de kNN. Una vez obtenido, el nuevo color y su composición pasa al catálogo de muestras. El desarrollo de

esta herramienta RBC ha permitido a GE Plastics ahorrar millones de dólares en costes de material y producción.

- **Call Center de Microsoft** en el país de Gales. Este es un claro ejemplo de aplicación de los SRBC en el campo del help-desk. El sistema se diseñó para ser utilizado por los operadores telefónicos que responden a las consultas de los clientes, recuperando consultas pasadas resueltas con éxito. En sólo 9 meses de funcionamiento se consiguió un aumento del 10 % en la satisfacción de los clientes, de un 28 % en el ratio de consultas resueltas en primera instancia, y de un 13 % en la consideración de los clientes sobre los conocimientos de los agentes. Además, se constató una reducción en el periodo de formación de nuevos agentes y una mejora considerable en la consistencia de las respuestas proporcionadas.

22.6 Herramientas para el desarrollo de SRBC

Como podemos apreciar en la sección anterior, son numerosos los dominios y casos de éxito industrial de los SRBC. Sin embargo, todos estos sistemas han tenido que solventar los mismos escollos metodológicos: identificación del problema a resolver (p.e.: diagnóstico, help-desk, planificación, etc.) y el desarrollo del SRBC. Una vez delimitado el problema que el sistema debe resolver, el desarrollo implica numerosas decisiones acerca del diseño, que afectan directamente a la eficiencia del sistema final. En la práctica, estas decisiones se pueden resumir en:

- Diseño de los elementos principales del SRBC: casos, LC, índices, similitud global, similitud local (véase la sección 22.3).
- ¿Qué etapas del ciclo RBC (véase la sección 22.4) se van a implementar? y ¿cómo se van a desarrollar?

Además, en cualquier desarrollo de este tipo, hay que tener en cuenta que los sistemas RBC tienen un proceso de aprendizaje mediante retroalimentación de la LC (casos adaptados de la etapa *Retener*), por lo que la vida de estos sistemas requiere cierto mantenimiento que debe considerarse. En la mayoría de las ocasiones, gran parte de estas decisiones de diseño implican la selección de una herramienta adecuada para el desarrollo del sistema. En este sentido, son varios los grupos de investigación que ofrecen herramientas para el desarrollo de SRBC de forma gratuita, y que en poco a poco van incorporando resultados de investigación. Entre las más destacables podemos citar:

- **FreeCBR¹**, Está diseñado como un motor para RBC y está implementado en Java. Está compuesto por una pequeña herramienta gráfica (véase la Figura 22.8), para la descripción y búsqueda de casos, un intérprete de comandos, facilidades para desarrollar aplicaciones web y una API de desarrollo. Los casos se

¹<http://freecbr.sourceforge.net/>

describen mediante un vector de atributos que pueden ser de tipo cadena, multievaluado, entero, real y booleano. Permite configurar la medida de similitud para la recuperación de casos. Esencialmente emplea una función de similitud global como la media ponderada de las similitudes locales de los atributos pudiendo seleccionar los pesos. La medida de similitud local para cada atributo es configurable, dando la oportunidad de utilizar funciones basadas en lógica borrosas (lineal y logarítmica) o concretas, operadores lógicos ($<$, $>$, $=$, \max , \min), etc. Inicialmente FreeCBR se limita a la implementación de búsquedas de los casos más similares, dejando a un lado el resto de etapas del ciclo RBC, que deberán ser implementadas a partir de la API que ofrece.

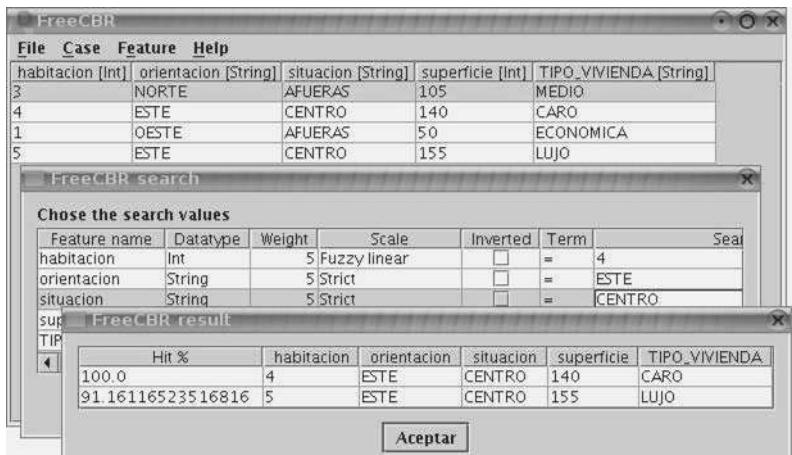


Figura 22.8: FreeCBR.

- **CBRshell²**, desarrollado por el Instituto de Aplicaciones de Inteligencia Artificial (AIAI) de Edimburgo ofrece un shell para RBC que permite la búsqueda de casos más similares y ofrece algunas funcionalidades de optimización del tiempo de ejecución mediante algoritmos genéticos (véase la Figura 22.9). Esta herramienta, disponible en Java y C++, permite la establecimiento de pesos entre los atributos y configuración de similitudes locales mediante distancias euclídeas (atributos numéricos) y correspondencia de caracteres (atributos de texto). También tiene un módulo que permite configurar los algoritmos genéticos para la optimización.

²<http://www.aiai.ed.ac.uk/technology/casebasedreasoning.html>



Figura 22.9: CBRshell.

- **CBR*Tools**³ desarrollado por el Instituto Francés de Investigación (INRIA), se presenta como un framework orientado a objetos (véase la Figura 22.10). Este framework facilita el análisis y el desarrollo de SRBC definiendo en UML (Unified Model Language) los elementos fundamentales de la gestión de un SRBC, definición de los casos y la organización de la memoria.

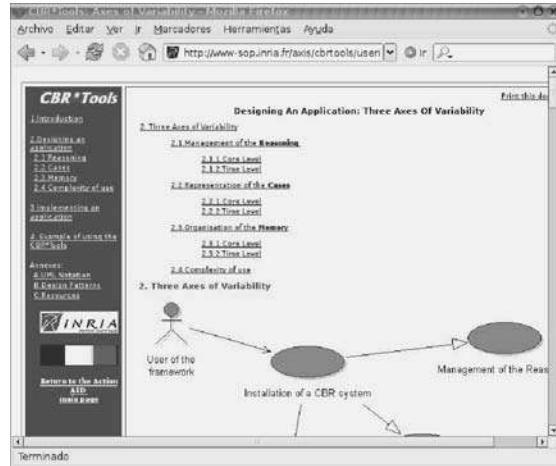


Figura 22.10: CBR*Tools.

³<http://www-sop.inria.fr/axis/cbrtools/>

- **jColibrí**⁴, es un desarrollo del Grupo de Aplicaciones de Inteligencia Artificial (GAIA) de la Universidad Complutense de Madrid. jColibrí también se distribuye como un framework orientado a objetos desarrollado en Java. Una de las principales características de jColibrí es que ofrece una aplicación parcialmente desarrollada y genérica que debe ser extendida e implementada para que sea aplicada a un dominio concreto (véase la Figura 22.11). A diferencia de otros frameworks, el desarrollo en jColibrí está enfocado desde el punto de vista de la Ingeniería del Conocimiento, proporcionando una infraestructura basada en tareas, subtareas y métodos modelada mediante la ontología CBROnto. Un ejemplo de ello es que, para las cuatro etapas del ciclo RBC, jColibrí ofrece una librería de métodos de resolución (PSM). Además, jColibrí permite la configuración de aplicaciones, basada en un motor de inferencia de Lógica Descriptiva, utilizando una interfaz gráfica que permite la selección de métodos para la resolución de las diferentes tareas.

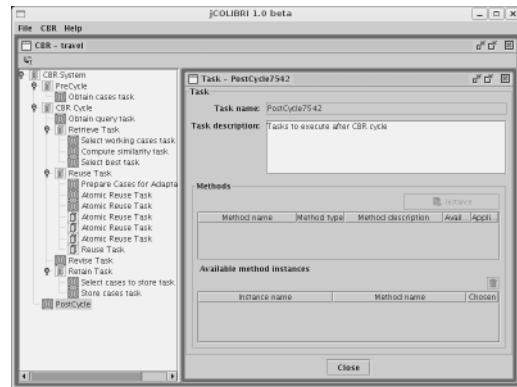


Figura 22.11: jColibrí.

Entre las herramientas comerciales que más impacto han tenido cabe destacar:

- **ESTEEM**⁵ es una herramienta comercializada por Stottle Henke Associates Inc. para el desarrollo de sistemas RBC para la toma de decisiones. Esta herramienta tiene una cómoda interfaz gráfica que permite la definición de casos, la definición de funciones de similitud, la edición de casos (para almacenar en la Librería de Casos) y un editor de reglas. Un caso está definido por un conjunto de atributos, sin embargo, una de las particularidades de ESTEEM es que, aparte de los tipos habituales (booleano, cadena, numérico), permite que existan casos como atributos de otros casos, pudiendo obtener casos anidados.

⁴<http://gaia.fdi.ucm.es/grupo/projects/jcolibri/>

⁵http://www.stottlerhenke.com/solutions/decision_support/esteem.htm

Además, soporta varios métodos para describir la función de similitud local (parcial, rangos, borrosas...) y para similitud global (pesos, conteo, inferencia de pesos automática...). Adicionalmente, en la LC se pueden establecer reglas IF-THEN tanto para establecer la similitud, como para formalizar el proceso de adaptación de casos.

- **Art*Enterprise**⁶ comercializado por la compañía Mindbox es un entorno de desarrollo de aplicaciones para la toma de decisiones basado en reglas y RBC. Este motor de RBC incluye un entorno que permite desarrollar aplicaciones programando directamente en C++ o en un lenguaje script propio (denominado ARTScript). Así, es posible definir nuevos casos, crear bases de casos, establecer consultas o describir el peso de los atributos. Además, es posible describir estructuras complejas de los casos. Por ejemplo, se pueden definir tipos de atributos como una jerarquía de elementos, atribuyéndole distancia entre ellos para medir similitudes. El entorno Art*Enterprise también incluye un completo control de versiones de desarrollo y recursos de integración y conectividad con diferentes bases de datos.
- **ReCall**⁷ es una herramienta de desarrollo de aplicaciones RBC y modelado del conocimiento comercializada por la empresa AliceSoft e implementada en C++. Permite una representación de casos orientada a objetos utilizando mecanismos de herencia y relaciones entre objetos, ofreciendo una interfaz visual para modelarlo. Esto permite un modelado del conocimiento muy flexible y potente. Los casos son introducidos en la LC como instancias de estas redes de objetos. Ofrece mecanismos de indexación jerárquicos para realizar recuperaciones eficientes. ReCall permite la generación automática de índices, pero también permite que sean modificados por los desarrolladores. La similitud entre casos está basada en *knn*. Implementa la etapa de *Adaptación* mediante dos modalidades: por un proceso de votación, o definido por el usuario mediante un conjunto de reglas.

22.7 Lecturas recomendadas y recursos en Internet

Para obtener una visión más amplia del campo del RBC se recomienda al lector examinar detenidamente estas fuentes bibliográficas:

- Aamodt, A., Plaza, E. (1994). **Case Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches**. AI Communications, 7, IOS Press, 39-59.
- Leake D. (ed.) (1996). **Case-Based Reasoning. Experiences, Lessons, & Future Directions**. Estados Unidos de América: AAAI Press.

⁶<http://www.mindbox.com/>

⁷http://www.alice-soft.com/html/prod_recall.htm

- Pal, S., Shiu S. (2004). **Foundations of Soft Case-Based Reasoning.** Estados Unidos de América: Wiley Interscience.
- Watson, I. (1997). **Applying Case-Based Reasoning: Techniques for Enterprise Systems.** Estados Unidos de América: Morgan Kaufmann.
- Watson I. (1998). **CBR is a methodology not a technology. Research & Development in Expert Systems**, Miles R., Moulton, M. & Bramer, M. (Eds.), Springer, 213-223.

Más información relativa a SRBCs experimentales tales como: manuales, cursos e información sobre la actividad de investigación que desarrolla la comunidad RBC (congresos, charlas, etc.), se pueden encontrar en las siguientes direcciones de Internet:

- **ai-cbr:** <http://www.ai-cbr.org/> (portal de investigación y recursos sobre RBC).
- **AAAI cbr:** <http://www.aaai.org/AITopics/html/casebased.html> (información sobre RBC de la Asociación Americana de Inteligencia Artificial).
- **IIIA rbc:** <http://www.iiia.csic.es/Projects/cbr/index.html> (investigación sobre RBC del Instituto de Investigación en Inteligencia Artificial (IIIA) del Consejo Superior de Investigaciones Científicas (CSIC) en España).
- **Home site E. Plaza:** <http://www.iiia.csic.es/People/enric.html> (recursos didácticos sobre RBC).
- **Home site D. Aha:** <http://home.earthlink.net/~dwaha/> (recursos didácticos sobre RBC).

22.8 Resumen

En este capítulo se ha analizado un nuevo paradigma para el diseño de sistemas inteligentes, el Razonamiento Basado en Casos (RBC). El elemento principal de este nuevo tipo de sistemas es el caso, que de alguna forma hace las veces de contenedor de la experiencia previa. Estos casos están organizados en una Librería de Casos (LC) que gracias a la utilización de mecanismos de indexación que permiten una búsqueda eficiente de los casos. Una vez se tiene LC, los usuarios pueden plantear nuevos problemas (casos nuevos). Para resolver un nuevo problema, el sistema busca en la LC aquéllos en los que la experiencia que representan se puede aplicar a la situación actual. Una vez se tiene ese conjunto de casos, se aplica alguna medida de similitud para determinar el caso más similar. Finalmente, este caso más similar es adaptado a la situación actual para intentar dar una solución. Si se considera oportuno, el nuevo caso resuelto puede ser incluido en la LC. En tal caso, estaríamos retroalimentando al sistema con su propia experiencia, dotándolo de cierta capacidad de aprendizaje. Este ciclo, se puede resumir en lo que se ha denominado *etapas-RE*:

Recuperar, Reutilizar, Revisar y Retener. Muchas de las justificaciones para este tipo de sistemas las podemos encontrar en cómo las personas resuelven problemas en la vida real. Generalmente, la segunda vez que intentamos resolver un problema es más fácil que la primera. Además, en la segunda resolución del problema seremos más eficientes, ya que podemos aprender de nuestros errores pasados y qué tenemos que hacer para no repetirlos. Una de las claves de los SRBC reside en su habilidad para aprender de su experiencia, de la misma forma que un médico recuerda un caso difícil de resolver, por si este se le presenta en un futuro. Como podemos ver, recordar situaciones pasadas es de gran utilidad para enfrentarnos a situaciones recurrentes. Además, recurrir a situaciones previas similares es necesario para poder tratar las complejidades inherentes a situaciones novedosas. Por lo tanto, recordar una situación pasada y utilizarla posteriormente para resolver un problema nuevo es una parte importante del proceso de aprendizaje.

1. Sistemas que, bien implementados, obtienen soluciones de una manera rápida.
2. No obtienen una solución inferiendo desde el principio del problema, sino que la adapta a partir de un problema resuelto ya conocido.
3. Son de fácil comprensión por parte del experto ya que el sistema no maneja conceptos abstractos, sino situaciones concretas (casos) del dominio conocido por el experto.
4. En la LC, cada caso se considera un bloque de conocimiento independiente del resto (al contrario que en los sistemas basados en reglas).
5. Los casos representan por igual situaciones típicas y excepcionales.
6. Son sistemas que con el tiempo mejoran su efectividad gracias a la incorporación de casos y tareas de mantenimiento.
7. El ciclo RBC permite un desarrollo estructurado del sistema.

Y entre sus desventajas cabe destacar:

1. Los casos son situaciones muy concretas y sus situaciones no son siempre adaptables a otros problemas.
2. La recuperación y adaptación de casos requiere, en ocasiones, un profundo conocimiento del dominio.
3. Requieren otras técnicas de la IA para obtener resultados.

22.9 Ejercicios resueltos

22.1. *La Agencia Inmobiliaria* ofrece un servicio abierto a sus clientes para orientarles en el precio de alquiler a la hora de buscar vivienda. Puesto que los precios de la vivienda varían cada año, se ha decidido que la información relativa a los precios sea

cualitativa, considerando que es la mejor estrategia para satisfacer las necesidades informativas de potenciales clientes. Para ello, en la Tabla 22.3 se dispone de la base de datos de las viviendas que actualmente oferta la agencia inmobiliaria para alquiler.

Orientación	Situación	Sup.	Dorm.	Baños	Precio
NORTE	AFUERAS	105	3	2	ECONÓMICO
ESTE	AFUERAS	140	3	1	CARO
OESTE	AFUERAS	50	2	1	MUY BARATO
ESTE	AFUERAS	155	5	3	LUJO
ESTE	AFUERAS	150	4	3	CARO
NORTE	AFUERAS	100	3	2	ESTÁNDAR
OESTE	BARRIO CERCANO	120	4	2	ESTÁNDAR
NORTE	BARRIO LEJANO	85	2	2	ECONÓMICO
NORTE	AFUERAS	50	1	1	CARO
SUR	AFUERAS	200	5	3	CARO
NORTE	BARRIO CERCANO	75	3	1	CARO
NORTE	BARRIO CERCANO	50	1	1	ECONÓMICO
OESTE	BARRIO LEJANO	100	2	2	ESTÁNDAR
OESTE	AFUERAS	65	2	1	MUY BARATO
NORTE	AFUERAS	100	3	1	CARO
OESTE	AFUERAS	80	2	1	ECONÓMICO
ESTE	AFUERAS	145	4	3	LUJO
ESTE	AFUERAS	170	4	3	MUY CARO
NORTE	AFUERAS	90	3	2	ESTANDAR
SUR	BARRIO CERCANO	110	4	2	ESTANDAR
NORTE	BARRIO LEJANO	65	2	1	MUY BARATO
NORTE	AFUERAS	45	1	1	ESTÁNDAR
SUR	AFUERAS	220	5	3	LUJO
SUR	BARRIO CERCANO	75	3	1	MUY CARO
NORTE	BARRIO CERCANO	40	1	1	MUY BARATO
OESTE	BARRIO LEJANO	120	2	2	ESTÁNDAR

Tabla 22.3: Librería de Casos de apartamentos de la Agencia Inmobiliaria.

El servicio ofertado consistirá en preguntarle al cliente cuáles son las características generales de la vivienda buscada, dando como respuesta el coste orientativo de la misma. Los criterios para el establecimiento del precio del alquilar dependen fundamentalmente de la situación y de la superficie, en menor grado del número de habitaciones y finalmente, en algunas ocasiones, de la orientación y del número de aseos.

Solución (con FreeCBR):

La herramienta gráfica que ofrece FreeCBR nos permitirá automatizar parte de este servicio realizando la búsqueda más similar entre los casos de su LC. Desde el punto de vista estricto no permite el desarrollo de SRBC, ya que no implementa el ciclo RBC completo. Por ejemplo, no establece medidas para la adaptación de soluciones (que deben ser modeladas como un atributo más del caso) o para la incorporación automática de nuevos casos (que deben ser introducidos manualmente como el resto). Sin embargo, esta herramienta ofrece facilidades para la definición de casos, almacenamiento en LC, y la edición/ejecución de consultas de los casos más similares.

En primer lugar describiremos un caso atendiendo a los tipos de datos que permite: orientación y situación son de tipo string; superficie, número de dormitorios y baño de tipo entero y el precio es un dominio de etiquetas cualitativas (MUY ECONÓMICO,

ECONÓMICO, ESTÁNDAR, CARO, MUY CARO, LUJO). Para ello, utilizaremos el editor de casos, añadiendo cada atributo y determinando su tipo.



Figura 22.12: Edición de atributos.

Una vez definido qué es un caso, se incorporará en la LC cada uno de las viviendas descritas en la Tabla 22.3.

orientacion [String]	situacion [String]	superficie [Int]	dormitorios [Int]	bños [Int]	PRECIO [MultiString]
NORTE	AFUERAS	105	3	2	ECONOMICO
ESTE	CENTRO	140	3	1	CARO
OESTE	AFUERAS	50	2	1	MUY ECONOMICO
ESTE	CENTRO	155	5	3	LUJO
ESTE	AFUERAS	150	4	3	CARO
NORTE	AFUERAS	100	3	2	ESTANDAR
OESTE	BARRIO CERCANO	120	4	2	ESTANDAR
NORTE	BARRIO LEJANO	85	2	2	ECONOMICO
NORTE	CENTRO	50	1	1	CARO
SUR	AFUERAS	200	5	3	CARO
NORTE	BARRIO CERCANO	75	3	1	CARO
NORTE	BARRIO CERCANO	50	1	1	ECONOMICO
OESTE	BARRIO LEJANO	100	2	2	ESTANDAR
OESTE	AFUERAS	65	2	1	MUY ECONOMICO
NORTE	CENTRO	100	3	1	CARO
OESTE	AFUERAS	80	2	1	ECONOMICO
ESTE	CENTRO	140	4	2	ECONOMICO

Figura 22.13: Librería de Casos en FreeCBR.

Para realizar la búsqueda del caso más similar, FreeCBR permite realizar consultas, estableciendo para cada una de ellas el tipo de similitud entre atributos (local) y los pesos para el cómputo de la similitud global (mediante la media ponderada). Por ejemplo, supongamos que el cliente quiere saber cuánto le podría costar el alquiler

de una vivienda en las afueras, de 120 metros cuadrados, cuatro dormitorios y dos cuartos de baño. Utilizando la nomenclatura utilizada en este capítulo:

$$\text{Caso}_{in} = ((4, 2, 120, \text{afueras}), (?)) \quad (22.18)$$



Figura 22.14: Función de similitud en FreeCBR.

The screenshot shows the 'FreeCBR result' window. It displays a table of house listings with columns: Hit %, orientacion, situacion, superficie, dormitorios, banos, and PRECIO. The table contains 30 rows of data. At the bottom is an 'Aceptar' button.

Hit %	orientacion	situacion	superficie	dormitorios	banos	PRECIO
88.42031258329375	ESTE	AFUERAS	150	4	3	CARO
85.12494357734437	ESTE	AFUERAS	170	4	3	MUY CARO
49.85080127778224	NORTE	AFUERAS	105	3	2	ECONOMICO
49.74089690682699	NORTE	AFUERAS	100	3	2	ESTANDAR
49.44345603673792	NORTE	AFUERAS	90	3	2	ESTANDAR
48.29382061195576	OESTE	AFUERAS	80	2	1	ECONOMICO
47.57927413608056	OESTE	AFUERAS	65	2	1	MUY ECONOMICO
46.72897951275032	OESTE	AFUERAS	50	2	1	MUY ECONOMICO
46.102004232176405	SUR	AFUERAS	200	5	3	CARO
44.73730714393525	SUR	AFUERAS	220	5	3	LUJO
44.09830056250526	OESTE	BARRIO CERCANO	120	4	2	ESTANDAR
44.03756893978645	SUR	BARRIO CERCANO	110	4	2	ESTANDAR
43.05410634841381	ESTE	CENTRO	145	4	3	LUJO
25.0	OESTE	BARRIO LEJANO	120	2	2	ESTANDAR
24.82701653033722	OESTE	BARRIO LEJANO	100	2	2	ESTANDAR
24.50444563699238	NORTE	BARRIO LEJANO	85	2	2	ECONOMICO
24.309165391507324	NORTE	CENTRO	100	3	1	CARO
24.309165391507324	ESTE	CENTRO	140	3	1	CARO
23.988792085786205	ESTE	CENTRO	155	5	3	LUJO
23.701727736883416	SUR	BARRIO CERCANO	75	3	1	MUY CARO
23.701727736883416	NORTE	BARRIO CERCANO	75	3	1	CARO
23.364939485244797	NORTE	BARRIO LEJANO	65	2	1	MUY ECONOMICO
22.780820881383523	NORTE	BARRIO CERCANO	50	1	1	ECONOMICO
22.780820881383523	NORTE	CENTRO	50	1	1	CARO
22.56791526483134	NORTE	CENTRO	45	1	1	ESTANDAR
22.346964336297003	NORTE	BARRIO CERCANO	40	1	1	MUY ECONOMICO

Figura 22.15: Resultado de búsqueda en FreeCBR.

Con este fin, realizaremos una consulta donde se determinan los criterios de similitud válidos para la búsqueda. En primer lugar, se deberá establecer el peso de cada uno de

los atributos en la consulta. Como se indicó en el enunciado la situación, superficie y dormitorios son los tres criterios más decisivos, mientras que la orientación y los aseos son menos relevantes. Por ello los tres primeros tienen un peso de 4 ó 5, mientras que los menos importantes tienen un peso de 2 (véase la Figura 22.14).

Una vez descritos estos pesos habrá que analizar una a una los métodos de similitud entre atributos. Por ejemplo, para medir la superficie se ha empleado una función borrosa logarítmica que permite aproximar de forma suave parámetros de superficie cercanos (una vivienda de 102 m^2 tiene un precio parecido a una de 105 m^2), mientras que el número de dormitorios es un valor estricto (no cuesta lo mismo una vivienda de 2 habitaciones que una de 3).

Finalmente realizaremos la consulta (véase la Figura 22.15), devolviendo los grados de similitud en orden decreciente (acuerdo o *Hit %*). Así, el resultado más similar (88.42 %) indica que el precio de alquiler es caro, que será el precio asignado a la consulta del cliente.

22.10 Ejercicios propuestos

22.1. En el sistema propuesto para la *Agencia Inmobiliaria*, la herramienta de desarrollo implementa una Memoria Plana. Describir, sobre papel un modelo de Memoria Jerárquica mediante discriminación de propiedades para los 10 primeros casos la LC (Tabla 22.3). ¿Qué criterios has adoptado para el establecer el orden de los nodos internos?

22.2. A partir del problema de la *Agencia Inmobiliaria*, desarrollar un modelo de Memoria Dinámica considerando los 10 primeros casos la LC (Tabla 22.3). Antes de comenzar, analizar cuál es el mejor criterio para establecer Episodios Generalizados y las Normas.

22.3. En el problema de la *Agencia Inmobiliaria*, estudiar para cada una de los atributos del caso, qué medida de similitud es más apropiada. Para ello hay que considerar, al menos, las medidas descritas en la Tabla 22.2.

22.4. Gracias a la herramienta FreeCBR, se ha desarrollado una versión muy simplificada de un SRBC. Describir qué tareas del Ciclo RBC se han desarrollado y cuáles no.

22.5. Establecer cómo desarrollaría la tarea *Reutilizar* para mejorar el sistema desarrollado para la *Agencia Inmobiliaria*. Analizar las diferentes alternativas indicando las ventajas e inconvenientes.

Referencias

- AAMODT, A. y PLAZA, E.: «Case-based reasoning : Foundational issues, methodological variations, and system approaches». *AI Communications*, 1994, **7(1)**, pp. 39–59.
- BAREISS, R.: *PROTOS; a unified approach to concept representation, classification and learning*. Tesis doctoral, University of Texas, University of Texas at Austin, USA, 1988.
- GOEL, A.: *Integration of case-based reasoning and model-based reasoning for adaptation*. Tesis doctoral, Ohio University, Ohio University, USA, 1989.
- HAMMOND, K.: «CHEF: a model of case-based planning». En: *Proceedings of the Fourth National Conference on Artificial Intelligence*, , 1986.
- HINRICHES., T. R.; y KOLODNER, J. L.: «The roles of adaptation in case-based design». En: *Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA, USA*, , 1991.
- KOLODNER, JANET L. y LEAKE, DAVID B.: «A Tutorial Introduction to Case-Based Reasoning». En: David B. Leake (Ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, capítulo 2, pp. 31–65. American Association for Artificial Intelligence, 1996.
- KOTON, P.: «Using experience in learning and problem solving». *Informe técnico MIT/LCS/TR-441*, MIT, 1989.
- LEAKE, DAVID B.: «CBR in Context: The Present and The Future». En: David B. Leake (Ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, capítulo 2, pp. 31–65. American Association for Artificial Intelligence, 1996.
- LÓPEZ, B. y PLAZA, E.: «Case-based learning of plans and goal states in medical diagnosis». *Artificial Intelligence in Medicine*, 1993, **9**, pp. 29–60.
- RIESBECK, C. K. y SCHANK, R. S.: *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, 1989.
- SCHANK, ROGER C.: *Dynamic Memory*. Cambridge University Press, 1982.
- SCHANK, ROGER C. y ABELSON, ROBERT P.: *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*. L. Erlbaum, Hillsdale, NJ, 1977.
- SIMPSON, R. L.: *A Computer Model of Case-based Reasoning in Problem-solving: An Investigation in the Domain of Dispute Mediation*. Tesis doctoral, School of Information and Computer Science, Georgia Institute of Technology, 1985.
- SYCARA, K.: «CADET: a case-based synthesis tool for engineering design.» *International Journal for Expert Systems*, 1992, **4(ii)**, pp. 157–188.

SYCARA, K. P.: *Resolving Adversarial Conflicts: An Approach to Integrating Case-Based and Analytic Methods*. Tesis doctoral, School of Information and Computer Science, Georgia Institute of Technology, 1987.

Capítulo 23

Reconocimiento de Formas

Mario Hernández Tejera, Javier Lorenzo Navarro

y Juan Méndez Rodríguez

Universidad de las Palmas de Gran Canaria

23.1 Introducción

Podemos definir de forma simplificada la disciplina de **Reconocimiento de Formas o Patrones** (*Pattern Recognition*) como aquella que concierne con el conjunto de procesos orientados a la transformación de datos en entidades con significado. Estos datos suelen corresponderse con la salida suministrada por un sistema de sensores que adquieren la información del mundo o entorno. Un ejemplo es la matriz de pixels suministrada por un sistema de adquisición de imágenes o el vector que corresponde a una señal muestrada por un sistema de adquisición de señales temporales. Sin embargo, más allá de este origen que podemos enmarcar como parte de la Percepción Artificial, el origen de los datos puede ser más abstracto y en el desarrollo de los métodos, técnicas y herramientas de la disciplina, no se explicita la naturaleza o semántica concreta de los datos. Por otro lado, las denominadas entidades con significado se refieren a las salidas capaces de desencadenar acciones o secuencias de actuaciones como, por ejemplo, una orden de rechazo en un sistema de control de calidad o bien una orden sobre un sistema robótico. En el caso más general podemos considerar a las entidades con significado como diagnóstico de un proceso de Toma de Decisiones (Decision Making).

El salto entre los datos (sensoriales o abstractos) y la interpretación de los mismos se efectúa por un proceso de Contrasteación o Clasificación. Dicho salto se suele efectuar a través del uso de una representación intermedia que permita superar el abismo semántico entre la estructura de los datos y la estructura de las interpretaciones.

Desde nuestro punto de vista, entendemos por **forma** o **patrón** cada una de las descripciones, cuantitativas o estructurales, de entidades o hechos del mundo o entorno del sistema. Del mismo modo, se puede definir una clase de formas como un conjunto de formas que poseen alguna propiedad común, y que pueden tener asociada alguna categoría semántica. El objetivo del Reconocimiento de Formas (RF) es, como hemos dicho, la asignación de formas a sus respectivas clases de manera mecanizada,

es decir, automática y por tanto, con la menor intervención humana posible. **Esto presupone que los datos admiten una categorización**, es decir, que existe algún tipo de regularidad entre ellos en función la cual se pueden definir las categorías o clases.

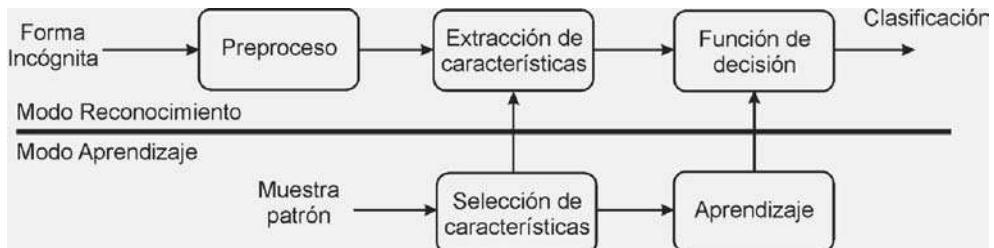


Figura 23.1: Diagrama de bloques de un sistema de Reconocimiento de Formas.

La Figura 23.1 muestra en forma de bloques los elementos fundamentales de un sistema de RF. Podemos observar la existencia de dos líneas de proceso de datos en interacción. La superior a la que simplificadamente podemos denominar como parte de la Teoría de la Decisión en formas más o menos complejas. La inferior es básicamente Aprendizaje Automático (*Machine Learning*). Si bien en sistemas avanzados ambas pueden ejecutarse en forma conjunta con aprendizaje continuo, lo normal en sistemas más convencionales es que la inferior sea de una tarea de aprendizaje off-line.

Podemos introducir un cierto grado de formalización para definir el problema de RF. Para ello, consideremos una situación como la mostrada en el diagrama de la Figura 23.2 que recoge las relaciones entre los conjuntos de datos involucrados en un sistema de RF. Sean:

- Σ el Dominio o Espacio Sensorial, es decir, el conjunto de cantidades o datos que pueden ser medidos por parte de los dispositivos del sistema sensorial.
- χ el Dominio o Espacio de las Representaciones intermedias o Características.
- Λ El conjunto de las interpretaciones o de categorías de Formas que se corresponde con las etiquetas lingüísticas asociadas al nivel de salida con cada una de las clases involucradas en el marco del sistema de reconocimiento de formas $\Lambda = \{I_i; i = 1, \dots, c\}$

En general, entre el dominio de las representaciones y el dominio de las interpretaciones se produce una compactación del volumen de la información asociada a cada forma. Además, cada interpretación I_i se suele hacer corresponder con un subconjunto del espacio o dominio de representación $\Omega_i \subseteq \chi$, más que con un solo elemento. El causante es el ruido que se introduce, por diferentes motivos, en los procesos del sistema sensorial y que provoca perturbaciones en los datos sensoriales. Ello, a su vez, se refleja en variaciones en las representaciones, por lo que una misma identidad lingüística o interpretación se podrá corresponder con diferentes representaciones que,

a su vez, se corresponderán con versiones “degradadas” de una hipotética representación ideal. De esta forma, existirán tantos subconjuntos $\{\Omega_i; i = 1, \dots, c\}$ en el espacio de representación como interpretaciones posibles. Cada uno de esos subconjuntos se corresponderá con una clase de formas, que agrupará a todas aquellas variaciones o versiones de la forma original, generadas por las causas mencionadas. Teniendo en cuenta lo anterior, podríamos definir una clase de formas como cada una de las c particiones del espacio de representaciones. El conjunto de las clases de formas cumplen, en general, las siguientes propiedades:

$$\Omega_i \neq \emptyset \quad \Omega_i \cap \Omega_j = \emptyset \quad \bigcup_{i=1}^c \Omega_i = \chi \quad (23.1)$$

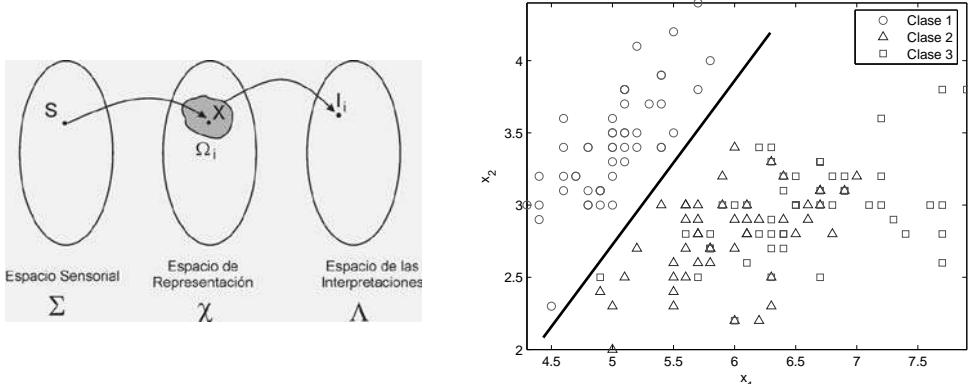


Figura 23.2: A la izquierda, relaciones entre los conjuntos de datos involucrados en un sistema de Reconocimiento de Formas. A la derecha, caso concreto de distribución de muestras de un problema de tres clases.

Es decir, no admitimos la interpretación nula o vacía, las interpretaciones son disjuntas y el conjunto de las mismas es completo. En este contexto, **un Clasificador o Regla de Clasificación** $d()$ es un proceso que tiene por objeto asignar una interpretación a cada representación, es decir: $d : \chi \rightarrow \Lambda$. Un clasificador $d()$ es pues un mecanismo sistemático de asignación de interpretaciones a datos sensoriales. La **partición** del espacio de representación según el conjunto de clases de formas Ω_i se efectúa mediante procesos de aprendizaje cuyo objetivo es determinar dicha partición a partir de las regularidades detectadas en las representaciones. Si el funcional de clasificación es tal que le asociamos una **Función de Confianza** $f \in [0, 1] \subseteq R$ cuyo objeto es asignar una medida de confianza a la interpretación I_i correspondiente a una representación \mathbf{X} :

$$d_f : \chi \rightarrow \langle \Lambda, F \rangle \quad (23.2)$$

El objetivo del Reconocimiento de Formas está relacionado con la construcción de algoritmos que implementen clasificadores $d()$ y de aquellos otros que permitan generarlos, es decir, algoritmos de aprendizaje. Aunque se suele olvidar con frecuencia, la correcta elección del espacio de representación suele aportar mayor potencia y posibilidades prácticas que la sofisticación de los procesos de clasificación y aprendizaje. En Reconocimiento de Formas es más útil una buena característica discriminante que la más sofisticada de todas las teorías; el esfuerzo debe realizarse prioritariamente en buscar las primeras. Esto se puede resumir mediante la receta metodológica siguiente: **Cuando se tengan dificultades en la clasificación, no intentemos resolver de entrada el problema a través de intrincadas herramientas matemáticas, sino, por el contrario y en lo posible, busquemos mejores características.**

Si bien en una primera presentación del problema de Reconocimiento de Formas hemos simplificado la naturaleza del problema, existen dos aproximaciones básicas a la solución de los problemas de RF más complejos, definidas en función del tipo de representación utilizada para las formas:

- **Aproximación de Teoría de la Decisión.** Se considera cuando el sistema manipula formas simples, entendiéndose por tales aquellas que al tiempo son los elementos primitivos de la representación, no planteándose descomposición estructural de las mismas. Se utilizan funciones de decisión para clasificar las formas, representadas como vectores de características.
- **Aproximación Estructural/Relacional.** Su utilidad se encuentra en problemas en los que las formas en estudio se consideran complejas, es decir, constituidas por formas simples y relaciones estructurales entre las formas simples. En este caso, las formas están representadas como cadenas, gramáticas, árboles, grafos, etc., estableciéndose una analogía entre la estructura de la forma y la sintaxis de un lenguaje, o entre una red estructural con contenido semántico. El proceso de reconocimiento resulta, en el caso sintáctico, uno de análisis (parsing), o en el caso estructural, un problema de comparación entre grafos o de inferencia.

En ambas aproximaciones, todo sistema de RF debe estar dotado de dos modos de funcionamiento: el denominado de reconocimiento propiamente dicho y el de análisis o aprendizaje. Las aproximaciones metodológicas al diseño de las etapas de aprendizaje pueden ser:

- **Aprendizaje Supervisado.** En este caso se dispone de un conjunto de muestras (**dataset**) controladas de las cuáles se conocen a priori la pertenencia a clase de todos y cada una de ellas. El proceso de aprendizaje consiste pues en la determinación de cuáles son las reglas de clasificación a partir de las regularidades de las muestras. En los capítulos 15 y 17, se pueden ver ejemplos de este tipo de técnicas.
- **Aprendizaje no Supervisado.** En este caso no se dispone de los valores de pertenencia a clase de los elementos de la muestra controlada, por lo que antes

de obtener las reglas de clasificación es preciso analizar el conjunto de datos para determinar el número de clases de formas que la constituyen, así como sus regularidades. La técnica del Análisis de Agrupamientos estudiada en el capítulo 16 es una forma muy común de este tipo de aprendizaje.

Un ejemplo notable de Aproximaciones Estructural/Relacional es el de interpretación del Lenguaje Natural, cuyo análisis requeriría un capítulo adicional, razón por la cual en este capítulo nos limitamos en presentar la Aproximación de Teoría de la Decisión, que se basa en la utilización de funciones de decisión para la clasificación de formas previamente representadas por vectores de características n -dimensionales: $\mathbf{X} = (x_1, \dots, x_n)^T$. Para tratar este problema, podemos utilizar varios esquemas que con generalidad podemos clasificar en los siguientes:

- **Esquema Geométrico.** En este caso, la clasificación se puede plantear como la asignación del vector de características incógnita, entendido este como un punto del espacio de características, a una de las particiones (regiones) mutuamente exclusivas previamente definidas en dicho espacio. Cada una de esas particiones se corresponde con una de las c clases de formas Ω_i . Para efectuar el proceso de asignación, durante el proceso de aprendizaje se define un conjunto de c funciones discriminantes $d_i(\mathbf{X})$, asociada cada una a una de las c clases de formas.
- **Esquema Estadístico.** En este caso se considera que los elementos del vector de características son variables aleatorias, siendo una medida ruidosa de la característica. Para cada clase de formas se hace uso del conocimiento de su distribución de probabilidad y de la probabilidad a priori de ocurrencia de cada clase. Basándose en estos datos se construye una regla de decisión de naturaleza probabilística basada, por ejemplo, en el objetivo de minimizar las probabilidades de reconocimiento erróneo.
- **Esquema de Redes Neuronales.** En este caso, se considera la utilización de elementos provenientes del paradigma de las Redes Neuronales Artificiales (ANN) (véase el capítulo 15), para resolver problemas de RF. Sin embargo, se puede considerar este esquema como uno independiente a los dos anteriores, o bien como una consecuencia del esquema geométrico.

23.2 Modelos estadísticos

En esta sección abordaremos el problema de la definición de reglas de decisión desde una aproximación estadística. En este caso, se considera a los vectores de características como variables aleatorias n -dimensionales y a las clases de formas como distribuciones de densidades de probabilidad [Devroye y otros, 1996; Fukunaga, 1990]. El objetivo será obtener reglas de clasificación óptimas en el sentido de minimizar determinadas tasas relacionadas con la clasificación errónea.

23.2.1 Clasificador bayesiano de mínimo error

Nos planteamos definir una regla de clasificación de formas entre clases Ω_i partiendo de un vector de medidas $\mathbf{X} \in \mathbf{U}$, conociendo las **probabilidades a priori**, $P(\Omega_i)$, de que una muestra pertenezca a una de las clases. Para dichas probabilidades se cumple:

$$\sum_{i=1}^c p(\Omega_i) = 1 \quad (23.3)$$

Supongamos que al diseñar un clasificador se disponen, o es posible estimar, las densidades de probabilidad de las clases: $p(\mathbf{X}/\Omega_i)$, que es la probabilidad condicional de una forma incógnita, sabiendo que pertenece a la clase Ω_i . La subfigura izquierda de la Figura 23.3 presenta un ejemplo de las densidades para un ejemplo unidimensional. Para un valor dado de \mathbf{X} , la diferencia entre estas densidades de probabilidad expresa la discriminabilidad entre las clases, siempre que \mathbf{X} haya sido seleccionada correctamente y contenga toda la información relevante al problema. Se debe verificar que:

$$\int_{\mathbf{U}} p(\mathbf{X}/\Omega_i) d\mathbf{X} = 1 \quad (23.4)$$

Sea además $p(\mathbf{X})$ la probabilidad de que el vector de características posea el valor \mathbf{X} , independientemente de su clase de pertenencia. En base a las probabilidades a priori, las densidades de probabilidad de clase y la probabilidad de que el vector posea un valor concreto, es posible determinar las **probabilidades a posteriori** $p(\Omega_i/\mathbf{X})$, es decir las probabilidades de que, poseyendo la forma incógnita a \mathbf{X} por valor de su vector de características, pertenezca a la clase Ω_i . Dicha determinación se puede realizar a partir del **Teorema de Bayes**:

$$p(\Omega_i/\mathbf{X}) = \frac{p(\mathbf{X}/\Omega_i)P(\Omega_i)}{p(\mathbf{X})} \quad (23.5)$$

donde se deduce que la probabilidad que figura en el denominador se puede expresar como:

$$p(\mathbf{X}) = \sum_{j=1}^c p(\mathbf{X}/\Omega_j)P(\Omega_j) \quad (23.6)$$

Debe verificarse, de forma análoga a la Ecuación 23.3, que: $\sum_{i=1}^c p(\Omega_i/\mathbf{X}) = 1$, y además: $\int_{\mathbf{U}} p(\Omega_i/\mathbf{X})p(\mathbf{X})d\mathbf{X} = 1$. La regla de clasificación estadística se puede establecer mediante la asignación de la forma incógnita \mathbf{X} a la clase con mayor probabilidad condicionada a la observación de la forma incógnita.

$$\mathbf{X} \in \Omega_i \leftrightarrow p(\Omega_i/\mathbf{X}) = \max_{j=1}^c p(\Omega_j/\mathbf{X}) \quad (23.7)$$

Dado que el término $p(\mathbf{X})$ es común a todas las clases, la regla de clasificación también puede expresarse como:

$$\mathbf{X} \in \Omega_i \leftrightarrow p(\mathbf{X}/\Omega_i)P(\Omega_i) = \max_{j=1}^c p(\mathbf{X}/\Omega_j)P(\Omega_j) \quad (23.8)$$

lo que es equivalente a introducir una función de decisión en la forma: $d_j(\mathbf{X}) = p(\mathbf{X}/\Omega_j)P(\Omega_j)$. En la subfigura derecha de la Figura 23.3 se utiliza este discriminante para efectuar la clasificación. Con bastante frecuencia se utiliza el logaritmo de este último término como función de decisión equivalente: $d'_j(\mathbf{X}) = \log p(\mathbf{X}/\Omega_j) + \log P(\Omega_j)$.

Podemos analizar a continuación las tasas de error en el clasificador probabilístico. La función de probabilidad condicional de error cometido para los valores de \mathbf{X} en el caso de dos clases será:

$$p(\epsilon/\mathbf{X}) = \begin{cases} p(\Omega_1/\mathbf{X}) & \mathbf{X} \in \Omega_2 \\ p(\Omega_2/\mathbf{X}) & \mathbf{X} \in \Omega_1 \end{cases} \quad (23.9)$$

Se observa cómo el error, en cada caso, viene definido por la menor de las probabilidades a posteriori para un \mathbf{X} dado. Por ello, a este esquema de clasificación se le denomina también **regla bayesiana de mínimo error**. La probabilidad de error total viene dada por:

$$p(\epsilon) = \int_{\mathbf{U}} p(\epsilon/\mathbf{X})p(\mathbf{X})d\mathbf{X} \quad (23.10)$$

Denominemos Γ_1 a la región del espacio donde se cumple que $p(\Omega_1/\mathbf{X}) > p(\Omega_2/\mathbf{X})$, es decir donde la forma incógnita se clasifica como Ω_1 , y Γ_2 a la región del espacio de representación donde se cumple que $p(\Omega_2/\mathbf{X}) > p(\Omega_1/\mathbf{X})$, es decir donde la forma incógnita se clasifica como Ω_2 . La probabilidad media de error que se comete al efectuar una clasificación es:

$$p(\epsilon) = \int_{\Gamma_1} p(\Omega_2/\mathbf{X})p(\mathbf{X})d\mathbf{X} + \int_{\Gamma_2} p(\Omega_1/\mathbf{X})p(\mathbf{X})d\mathbf{X} \quad (23.11)$$

Esta expresión puede transformarse en base al Teorema de Bayes en:

$$p(\epsilon) = p(\Omega_2) \int_{\Gamma_1} p(\mathbf{X}/\Omega_2)d\mathbf{X} + p(\Omega_1) \int_{\Gamma_2} p(\mathbf{X}/\Omega_1)d\mathbf{X} \quad (23.12)$$

$$p(\epsilon) = p(\Omega_2)E_{2 \rightarrow 1} + p(\Omega_1)E_{1 \rightarrow 2} \quad (23.13)$$

donde $E_{2 \rightarrow 1}$ es la probabilidad de que siendo la clase Ω_2 se clasifique como la clase Ω_1 y recíprocamente para $E_{1 \rightarrow 2}$. La probabilidad promedio de error consta de dos términos: el primero correspondiente a la clasificación errónea de muestras de Ω_2 en la región donde se clasifican como Ω_1 , y el segundo correspondiente a la clasificación errónea de muestras de Ω_1 en la región en la que se clasifican como Ω_2 . Es decir, podemos formar una matriz no simétrica con probabilidad de que una forma que sea Ω_i sea clasificada como Ω_j . El análisis anterior del clasificador bayesiano de mínimo error para el caso biclásico se puede extender fácilmente al caso multiclásico. Así, si el problema que se plantea es clasificar una muestra incógnita entre c clases, la regla puede escribirse, según la Expresión 23.8, a partir de las probabilidades a posteriori.

La probabilidad de error, que será mínima para cada decisión como hemos visto, de asignar \mathbf{X} a la clase Ω_i , será: $p_i(\epsilon/\mathbf{X}) = \sum_{j \neq i} p(\Omega_j/\mathbf{X}) = 1 - p(\Omega_i/\mathbf{X})$. Según las cuales, la regla de máxima probabilidad a posteriori, de la Ecuación 23.8, puede expresarse en la siguiente manera en forma de regla de clasificación de mínimo error:

$$\mathbf{X} \in \Omega_i \leftrightarrow p_i(\epsilon/\mathbf{X}) = \min_{j=1}^c p_j(\epsilon/\mathbf{X}) \quad (23.14)$$

El cálculo del error puede generalizarse para c clases como:

$$p(\epsilon) = \int_{\mathbf{U}} p(\epsilon/\mathbf{X})p(\mathbf{X})d\mathbf{X} = \sum_{i=1}^c \int_{\Gamma_i} p(\epsilon/\mathbf{X})p(\mathbf{X})d\mathbf{X} = \sum_{i=1}^c \sum_{j \neq i} \int_{\Gamma_i} p(\Omega_j/\mathbf{X})p(\mathbf{X})d\mathbf{X} \quad (23.15)$$

Si denominamos: $E_{j \rightarrow i} = \int_{\Gamma_i} p(\mathbf{X}/\Omega_j)d\mathbf{X}$, que es un elemento de una matriz de dimensión $c \times c$, no simétrica, a la que podemos denominar **Matriz de Confusión** que refleja la contribución al error de clasificar una forma Ω_j como si fuese Ω_i . El error total será:

$$p(\epsilon) = \sum_{i=1}^c \sum_{j \neq i} p(\Omega_j)E_{j \rightarrow i} \quad (23.16)$$

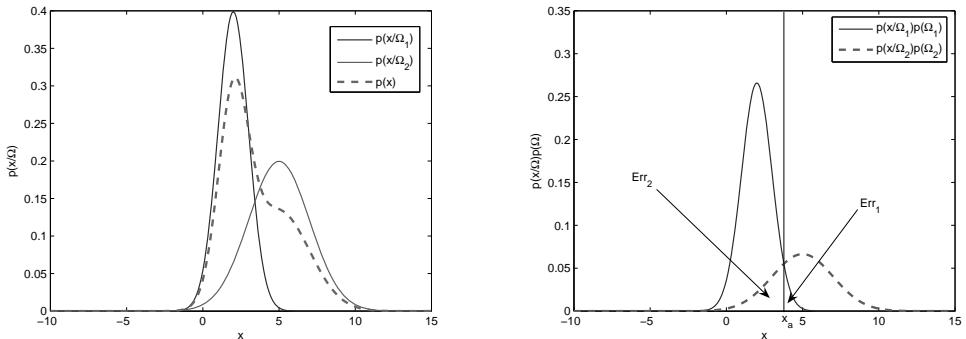


Figura 23.3: Ejemplos de Clasificador bayesiano de mínimo error en un caso unidimensional con dos distribuciones normales. El criterio de decisión se establece basándose en un valor x_a

23.2.2 Clasificador bayesiano de mínimo riesgo

Si bien una justificación teórica del Clasificador Bayesiano de Mínimo Riesgo requiere conceptos de Teoría de Juegos, podemos realizar una introducción cualitativa, no rigurosa, pero quizás más ilustrativa. El clasificador Bayesiano de Mínimo Riesgo supone que todos los errores son idénticos en importancia y por tanto se suman de

la misma forma, intentando minimizar el global. El clasificador Bayesiano de Mínimo Riesgo surge en situaciones en las que los errores no se pueden agrupar homogéneamente, sino que son diferentes en relevancia o importancia para el sujeto o sistema que los sufre.

Como ejemplo ilustrativo de la diferencia de aproximaciones, consideremos el caso de la toma de decisiones referidas al comportamiento de un sujeto en un entorno ciertamente peligroso como puede ser la sabana africana. En términos de Clasificador Bayesiano de Mínimo Error puede tener la misma trascendencia tanto el clasificar un observable que sea un león como si fuese un antílope, como el caso contrario, clasificar un antílope como un león. En términos del Clasificador Bayesiano de Mínimo Riesgo, ambos errores son diferentes y comportan riesgos diferentes. En resumen, **no todos los errores son idénticos ni comportan las mismas consecuencias para el sistema de decisión**. La función objetivo que se desea optimizar no debe ser homogénea en el tratamiento de los errores, para ello se debe introducir una nueva función objetivo que sea sustancialmente heterogénea y que permita introducir una valoración o intención propia del usuario y por tanto ajena a la estadística subyacente.

Para modelar esta nueva función introduciremos el concepto de pérdida, castigo o penalización por adoptar una decisión errónea. Sea que \mathbf{X} pertenece a la clase Ω_i , si el clasificador decide que pertenece a la clase Ω_j , incurre en una pérdida o penalización L_{ij} . Ahora bien, como existen c clases, el vector incógnita puede pertenecer a cualquiera de ellas, por lo que la pérdida promedio esperada o riesgo de asignar la muestra incógnita a la clase Ω_j será:

$$r_j(\mathbf{X}) = \sum_{i=1}^c L_{ij} p(\Omega_i / \mathbf{X}) \quad (23.17)$$

Aplicando el Teorema de Bayes y dado que $p(\mathbf{X})$ es un término común:

$$r_j(\mathbf{X}) = \frac{1}{p(\mathbf{X})} \sum_{i=1}^c L_{ij} p(\mathbf{X} / \Omega_i) p(\Omega_i) \quad (23.18)$$

La regla del Clasificador Bayesiano de Mínimo Riesgo puede expresarse como:

$$\mathbf{X} \in \Omega_i \leftrightarrow r_i(\mathbf{X}) = \min_{j=1}^c r_j(\mathbf{X}) \quad (23.19)$$

A efectos prácticos el término común $p(\mathbf{X})$ puede eliminarse para realizar las comparaciones. En algunos problemas de Reconocimiento de Formas, la pérdida es nula para decisiones correctas, y un valor fijo distinto de cero para decisiones erróneas. Así, por ejemplo, podemos definir: $L_{ij} = 1 - \delta_{ij}$, donde δ_{ij} representa la delta de Kronecker. A esta función se la denomina función de pérdida simétrica o cero-uno. Sustituyendo la expresión anterior en la del riesgo esperado se obtiene:

$$r_j(\mathbf{X}) = \sum_{i=1}^c (1 - \delta_{ij}) p(\Omega_i / \mathbf{X}) = 1 - p(\Omega_j / \mathbf{X}) \quad (23.20)$$

En este caso, la minimización del término de la izquierda, $r_j(\mathbf{X})$, implica la maximización del término $p(\Omega_j/\mathbf{X})$, lo que implica que el clasificador de mínimo riesgo se transforma en el de máxima probabilidad a posteriori o clasificador de mínimo error. Este resultado es lógico dado que la función de pérdida propuesta es tal que todos los errores se ponderan con el mismo valor, y por tanto nos encontramos en el problema ya considerado previamente.

23.2.3 Distribuciones normales multivariantes

La estructura de los clasificadores bayesianos está determinada, en principio, por la forma de las densidades condicionales $p(\mathbf{X}/\Omega_j)$. De las diferentes funciones estudiadas, ninguna ha recibido tanta atención como la densidad normal multivariante, fundamentalmente debido a su tratabilidad analítica. La densidad de probabilidad normal multivariante tiene la forma:

$$p(\mathbf{X}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})} \quad (23.21)$$

siendo \mathbf{X} un vector de características n -dimensionales, $\boldsymbol{\mu}$ el vector media y Σ la matriz de covarianzas de variables, de dimensión $n \times n$. Análogamente al caso univariante, la densidad normal multivariante se suele representar en forma reducida como $N(\boldsymbol{\mu}, \Sigma)$, de modo similar al caso unidimensional $N(\mu, \sigma)$, y está completamente definida por $n + n(n+1)/2$ parámetros que son: los elementos del vector de medias y los elementos independientes de la matriz de covarianzas, que es una matriz simétrica y definida positiva.

Abordamos seguidamente el diseño y análisis de un clasificador Bayesiano de mínimo error en un problema multiclásico (c clases) y multivariante (n -dimensional). Sea que las probabilidades a priori de las clases son $p(\mathbf{X}/\Omega_i)$ conocidas, y que las densidades de probabilidad de las mismas se rigen por ley normal, es decir:

$$p(\mathbf{X}/\Omega_i) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)} \quad (23.22)$$

Dada la naturaleza exponencial de la función de densidad, podemos definir una función discriminante asociada a cada clase como el logaritmo de su distribución de probabilidad como expresamos anteriormente. Eliminando los términos que son comunes a todas las clases, que no dependen del índice i , se tiene:

$$d_i(\mathbf{X}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i) - \frac{1}{2} \log |\Sigma_i| + \log P(\Omega_i) \quad (23.23)$$

Esta función discriminante es de origen probabilístico, pero se reduce a una de tipo geométrico de naturaleza cuadrática. Esta expresión puede desarrollarse resultando un discriminante cuadrático en la forma siguiente:

$$d_i(\mathbf{X}) = \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \omega_i^T \mathbf{x} + \omega_{i0} \quad (23.24)$$

Donde el término cuadrático es: $\mathbf{W}_i = -\frac{1}{2}\Sigma_i^{-1}$, el lineal: $\omega_i = \Sigma_i^{-1}\boldsymbol{\mu}_i$ y finalmente el independiente es: $\omega_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^T \Sigma_i^{-1} \boldsymbol{\mu}_i - \frac{1}{2} \log |\Sigma_i| + \log P(\Omega_i)$.

23.2.4 Estimación de distribuciones

Los métodos de clasificación estadísticos son de entre todos los propuestos los más rigurosos desde un punto de vista formal y son de uso general. Su punto débil es que precisan un conocimiento preciso de las distribuciones de probabilidad de las clases. En la práctica los datos que se disponen para construir los clasificadores son conjuntos más o menos extensos de muestras a partir de los cuales no siempre es posible estimar fiablemente las distribuciones de probabilidad. Consideraremos tan sólo brevemente el tema de la estimación de distribuciones que es imposible tratar en su totalidad en este libro. Las técnicas de estimación pueden clasificarse en función del conocimiento a priori disponible o asumido acerca de la naturaleza de las clases. En primer lugar, tenemos la **estimación paramétrica**, aplicable cuando se conoce la forma analítica de las funciones de distribución de las muestras pero no así sus parámetros. En segundo lugar, la **estimación no paramétrica** cuando no se conoce tal naturaleza, en la que en que intentaremos aproximar una distribución de probabilidad consistente con las muestras.

En la estimación paramétrica conocemos las funciones de distribución de cada clase: $p(\mathbf{X}/\Omega_i, \boldsymbol{\theta})$ donde $\boldsymbol{\theta}$ es un conjunto de parámetros que definen la función pero que son desconocidos. El objetivo es encontrar aquellos valores que son consistentes con las muestras disponibles. Dado que el procedimiento es indistinto de la clase, de forma simplificada utilizaremos la función $p(\mathbf{X}, \boldsymbol{\theta})$. Una de las técnicas básicas de estimación paramétrica es la de **Máxima Verosimilitud** (Maximum-Likelihood, ML). Dado un conjunto de datos D con m muestras \mathbf{X}_j pertenecientes a una clase concreta, la probabilidad de que tal muestra sea obtenida en un muestreo es $p(\mathbf{X}_j, \boldsymbol{\theta})$. Dado que las muestras son independientes unas de otras, la probabilidad de obtener el conjunto de datos completo será:

$$p(D/\boldsymbol{\theta}) = \prod_{j=1}^m p(\mathbf{X}_j, \boldsymbol{\theta}) \quad (23.25)$$

donde $p(D/\boldsymbol{\theta})$ es la probabilidad de obtener un conjunto de datos condicionada a un conjunto de parámetros. El conjunto de parámetros que consideraremos como solución $\hat{\boldsymbol{\theta}}$ será aquel en el que sea máxima la probabilidad de observar el conjunto de datos que realmente tenemos. De forma cualitativa, no rigurosa, podemos definirlo como: de todas las realidades posibles, la que observamos es la más probable. La solución será:

$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} p(D/\boldsymbol{\theta}) \quad (23.26)$$

Podemos utilizar una función objetivo más sencilla computacionalmente, como el logaritmo de la anterior $l(\boldsymbol{\theta}) = \ln p(D/\boldsymbol{\theta}) = \sum_{j=1}^m \ln p(\mathbf{X}_j, \boldsymbol{\theta})$. El problema de estimación paramétrica se puede resolver mediante la utilización del método del gradiente que precisa el gradiente en el espacio de los parámetros, bien: $\nabla_{\boldsymbol{\theta}} p(D/\boldsymbol{\theta})$ o bien: $\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$. Si conocemos la distribución de probabilidad a priori de los parámetros, $p(\boldsymbol{\theta})$, podemos utilizar la estimación Bayesiana para obtener la probabilidad $p(\boldsymbol{\theta}/D)$, que es

la que tiene mayor interés pues nos permite computar la distribución de probabilidad condicionada a la observación del conjunto de datos: $p(\mathbf{X}/D)$.

$$p(\boldsymbol{\theta}/D) = \frac{p(D/\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(D/\boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \quad p(\mathbf{X}/D) = \int p(\mathbf{X}/\boldsymbol{\theta})p(\boldsymbol{\theta}/D)d\boldsymbol{\theta} \quad (23.27)$$

donde $p(\mathbf{X}/\boldsymbol{\theta})$ es la forma analítica de la distribución de probabilidad. Otra modalidad más avanzada es la estimación basada en la Maximización de Expectativas (Expectation-Maximization EM) [Duda y otros, 2001, sec. 3.9][Theodoridis y Koutroumbas, 2003, pág. 36][Web, 2002, sec. 2.3] que incluye la posibilidad de conocimiento parcial de las coordenadas de las muestras, es decir con valores perdidos de los atributos.

En cuanto a la estimación no paramétrica, su objetivo es obtener las funciones de distribución de probabilidad sin suponer ninguna forma analítica concreta. Una de las maneras más simples de obtener la distribución de probabilidad se basa en el clasificador de los k vecinos más próximos (k NN) que se presenta en los procedimientos geométricos de clasificación. En este caso para cada punto del espacio de características se obtienen las k muestras del conjunto de datos más próximas al mismo. Si n_i es el número de las mismas que pertenecen a la clase Ω_i , entonces una posible estimación de las probabilidades es:

$$p(\Omega_i/\mathbf{X}) = \frac{n_i}{k} \quad \sum_{i=1}^c n_i = k \quad (23.28)$$

Otra manera más sencilla de construir las distribuciones de probabilidad es utilizando Métodos de Núcleos (*Kernel Methods*), donde la probabilidad se obtiene en cada punto por la contribución de una función de núcleo centrada en cada muestra. Para simplificar consideraremos el caso de una clase unidimensional con m muestras de la misma, de tal forma que:

$$p(x/\Omega) = \frac{1}{m h(m)} \sum_{j=1}^m \Psi\left(\frac{x - x_j}{h(m)}\right) \quad (23.29)$$

La función $\Psi(u)$, que constituye un núcleo, puede adoptar diversas formas y el parámetro $h(m)$ depende del número de muestras. Para que los estadísticos de la distribución continua $p(x/\Omega)$ coincidan total o asintóticamente con los del conjunto finito de muestras, se deben verificar algunas condiciones relacionadas con que ha de ser finita y asintóticamente nula:

$$\int \Psi(u)du = 1 \quad \int |\Psi(u)|du < \infty \quad (23.30)$$

$$\sup_u |\Psi(u)| < \infty \quad \lim_{u \rightarrow \infty} |u\Psi(u)| = 0 \quad (23.31)$$

Para que no existan desviaciones de las estimas, debe verificarse:

$$\lim_{m \rightarrow \infty} h(m) = 0 \quad \lim_{m \rightarrow \infty} m h(m) = \infty \quad (23.32)$$

Por ejemplo, una elección del tipo: $h(m) = h_0/\sqrt{m}$ verifica este criterio. A la hora de diseñar un estimador de distribución es necesario elegir un tipo de núcleo y el parámetro h_0 . Uno de los métodos de núcleos más comunes es el de las **Ventanas de Parzen** que utiliza una gaussiana para la función del núcleo. La elección del parámetro suele realizarse mediante el entrenamiento de un clasificador, de tal forma que su valor sea tal que genere la menor tasa de error del clasificador.

23.3 Modelos geométricos de la decisión

El objetivo de esta sección es el estudio de las reglas de decisión según la aproximación Geométrica de la Teoría de la Decisión. Si bien las Reglas de Decisión basadas en planteamiento estadístico son mucho más rigurosas que los planteamientos geométricos, estos últimos son más intuitivos y resultan suficientemente precisos en muchas situaciones.

23.3.1 Funciones discriminantes

Dado un espacio de representación donde se ha definido un conjunto de c clases $\{\Omega_1, \Omega_2, \dots, \Omega_c\}$, asociada a cada clase se define un funcional $d_i(\mathbf{X})$, donde $\mathbf{X} \in \mathbf{R}^n$ representa el vector de características del espacio. Se puede establecer una **Regla de Clasificación** basada en estos funcionales de la siguiente forma:

$$\mathbf{X} \in \Omega_i \leftrightarrow d_i(\mathbf{X}) = \max_j(d_j(\mathbf{X})) \quad (23.33)$$

Al conjunto de funcionales mencionados se les denomina **Funciones de Decisión o Funciones Discriminantes**. La siguiente expresión: $d_{ij}(\mathbf{X}) = d_i(\mathbf{X}) - d_j(\mathbf{X}) = 0$ en el espacio n -dimensional se corresponde con una hipersuperficie que separa las clases Ω_i y Ω_j . Esta hipersuperficie se denomina **Frontera de Decisión**. El éxito de los esquemas de clasificación de formas mediante funciones de decisión depende de dos factores. En primer lugar, de la forma de la función de decisión que debe estar directamente relacionada con el comportamiento geométrico de las clases en consideración. En segundo lugar, de la determinación de los parámetros de la función, que se resuelve mediante esquemas de aprendizaje.

23.3.1.1 Discriminante lineal

La lineal es la más sencilla de todas la posibles expresiones matemáticas de las funciones de decisión. En este caso, una función discriminante lineal para la clase Ω_i puede expresarse como:

$$d_i(\mathbf{X}) = \omega_{0i} + \boldsymbol{\omega}_i^T \mathbf{X} \quad (23.34)$$

La frontera de decisión entre dos formas Ω_i y Ω_j es: $d_{ij}(\mathbf{X}) = d_i(\mathbf{X}) - d_j(\mathbf{X}) = 0$ que igualmente posee estructura lineal: $w_{0i} - w_{0j} + (\boldsymbol{\omega}_i^T - \boldsymbol{\omega}_j^T)\mathbf{X} = 0$. Si introducimos el término independiente dado por: $w_{0ij} = w_{0i} - w_{0j}$ y el vector: $\mathbf{w}_{ij} = \boldsymbol{\omega}_i - \boldsymbol{\omega}_j$ la expresión de la frontera de decisión entre las formas Ω_i y Ω_j pasa a ser:

$$w_{0ij} + \mathbf{w}_{ij}^T \mathbf{X} = 0 \quad (23.35)$$

que representa la ecuación de una estructura plana en el espacio n -dimensional, correspondiendo a una recta en el espacio de dimensión 2, a un plano para dimensión 3 y en general a un hiperplano para dimensiones superiores. Esta estructura plana divide el espacio total en dos regiones. Los puntos en los que se verifica $d_{ij}(\mathbf{X}) > 0$ se encuentran en la región del espacio más cercano a Ω_i mientras los que verifican $d_{ij}(\mathbf{X}) < 0$ se encuentran más cercanos a la forma Ω_j . En un sistema binario, con dos clases, este criterio puede reducirse a un criterio más simple basado en el signo de una expresión discriminante lineal mediante la introducción de una máquina lineal en la forma siguiente:

$$d(\mathbf{X}) = f(w_0 + \mathbf{w}^T \mathbf{X}) \quad (23.36)$$

donde $f(L(\mathbf{X}))$ es una función no lineal de tipo todo/nada que tiene como argumento una ecuación lineal $L(\mathbf{X}) = w_0 + \mathbf{w}^T \mathbf{X}$, la función no lineal todo/nada admite diversas versiones matemáticas (continuas o discontinuas), pero las más sencillas de tipo booleano son como:

$$f(u) = \begin{cases} +1 & u \geq 0 \\ -1 & u < 0 \end{cases} \quad f(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases} \quad (23.37)$$

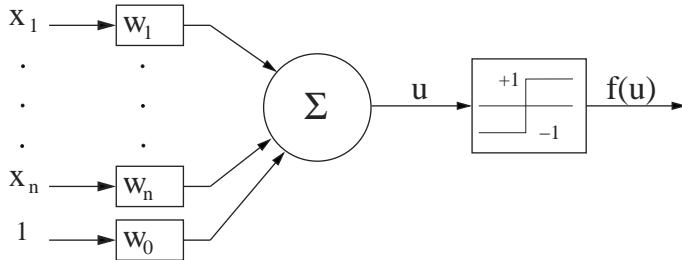


Figura 23.4: Modelo gráfico de la máquina lineal con coordenadas homogéneas. El modelo de función de activación $f(u)$ puede ser discreto como el mostrado, o bien continuo como las funciones sigmoides mostradas en la Figura 23.6.

La Figura 23.4 presenta una representación gráfica de la máquina lineal. Similarmente a otras disciplinas, suele ser más conveniente introducir una homogeneización de los sistemas lineales introduciendo las coordenadas homogéneas \mathbf{Y} definidas como:

$$\mathbf{Y} = \begin{pmatrix} 1 \\ \mathbf{X} \end{pmatrix} \quad \mathbf{a} = \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix} \quad (23.38)$$

donde podemos considerar una dimensión 0, de tal forma que: $y_0 = 1$ y $a_0 = w_0$. Las expresiones relativas a discriminantes lineales pueden expresarse en coordenadas homogéneas en la forma siguiente:

$$w_0 + \mathbf{w}^T \mathbf{X} = \mathbf{a}^T \mathbf{Y} \quad (23.39)$$

Para finalizar con el clasificador lineal, podemos adoptar una representación simplificada para conjuntos de datos con dos clases o binarios. Sea $y_i \in \{-1, +1\}$ una etiqueta de la clase a la que pertenece. El discriminante o clasificador lineal puede representarse matemáticamente como:

$$w^T \mathbf{X}_i + w_0 > 0 \quad y_i > 0 \quad (23.40)$$

$$w^T \mathbf{X}_i + w_0 < 0 \quad y_i < 0 \quad (23.41)$$

Ambas ecuaciones pueden simplificarse en una sola:

$$y_i(w^T \mathbf{X}_i + w_0) > 0 \quad (23.42)$$

La extraordinaria potencia del discriminante lineal reside en su simplicidad computacional, pero su debilidad es que no permite separar clases cuando la frontera de decisión entre las mismas es más compleja, como la existente entre las clases Ω_2 y Ω_3 en la Figura 23.2. En estos casos, una primera alternativa puede consistir en elevar el nivel de complejidad del discriminante mediante la utilización de polinomios de mayor grado o bien la introducción de discriminantes más ricos en contenido semántico como los basados en medidas de distancia.

23.3.2 Clasificación por funciones de distancia

La manera más simple e intuitiva de realizar un proceso de clasificación de formas es probablemente la basada en los conceptos de distancia entre las formas y prototipos. El motivo de ello es que la proximidad o similaridad entre vectores de características es una manera obvia de determinar pertenencia a categorías. En este caso, los vectores de características se consideran como puntos de un espacio con estructura tal que admite la definición de una distancia e incluso si es posible de una métrica.

Definiremos el concepto formal de distancia, que no siempre coincide con el concepto intuitivo de la misma. Para ello, sea $\mathbf{U} \in \mathbf{R}^n$ un conjunto, finito o infinito, de elementos, que se corresponden con los vectores de características asociados a las muestras de un problema dado. Se denomina función de distancia D a una correspondencia $D : \mathbf{U} \times \mathbf{U} \rightarrow \mathbf{R}$. Para que D sea un funcional de distancia se debe cumplir que, para un par arbitrario $\mathbf{X}, \mathbf{Y} \in \mathbf{U}$:

$$D(\mathbf{X}, \mathbf{Y}) \geq D_0 \quad (23.43)$$

$$D(\mathbf{X}, \mathbf{X}) = D_0 \quad (23.44)$$

$$D(\mathbf{X}, \mathbf{Y}) = D(\mathbf{Y}, \mathbf{X}) \quad (23.45)$$

donde D_0 es un número real finito arbitrario. La primera propiedad indica que la función de distancia posee una cota inferior, la segunda que la distancia es mínima para el caso de elementos idénticos, y la tercera es la propiedad de simetría. La función de distancia se dice que además es una **métrica** si:

$$D(\mathbf{X}, \mathbf{Y}) = D_0 \rightarrow X = Y \quad (23.46)$$

$$D(\mathbf{X}, \mathbf{Z}) \leq D(\mathbf{X}, \mathbf{Y}) + D(\mathbf{Y}, \mathbf{Z}) \quad (23.47)$$

La última propiedad corresponde a la denominada desigualdad triangular. Los funcionales de distancia, por su definición, asignan valor numérico a la noción de disimilaridad o lejanía entre dos formas representadas por vectores de características, de manera que una gran similaridad o semejanza entre ambas formas se refleja en un valor pequeño del funcional de distancia. Si además $D_0 = 0$, se tiene el concepto de métrica del análisis funcional; pero si no es así, siempre se puede redefinir la distancia como: $D(\mathbf{X}, \mathbf{Y}) = D(\mathbf{X}, \mathbf{Y}) - D_0$ que evita el problema de la cota inferior no nula. Existen múltiples propuestas para generar funcionales específicos que cumplan con las propiedades de distancias. En Reconocimiento de Formas son de interés dos modalidades; la primera está relacionada con la definición de normas generalizadas de tipo L_p . Así, si $\mathbf{X} \in \mathbf{R}^n$ se define recíprocamente la norma L_p de \mathbf{X} y la distancia entre dos formas como:

$$\|\mathbf{X}\|_p = \left[\sum_{i=1}^n |x_i|^p \right]^{\frac{1}{p}} \quad D_p(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\|_p \quad (23.48)$$

De todos los valores posibles del parámetro p entero, los más útiles en Reconocimiento de Formas son: la distancia Manhattan (también conocida como *city blocks* o *taxicab*), que se corresponde con caso $p = 1$, la distancia Euclídea que se corresponde con el caso $p = 2$ y la distancia de Tablero de Ajedrez o Chebychev, que se corresponde con el caso $p = \infty$. La segunda posibilidad para generar funcionales de distancias es la basada en la utilización de distancias generadas por matrices simétricas inductoras de formas cuadráticas. En este caso, sea \mathbf{B} una de tales matrices que genera una norma y distancia correspondiente como:

$$\|\mathbf{X}\|_B = \sqrt{\mathbf{X}^T \mathbf{B} \mathbf{X}} \quad D_B(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\|_B = \sqrt{(\mathbf{X} - \mathbf{Y})^T \mathbf{B} (\mathbf{X} - \mathbf{Y})} \quad (23.49)$$

Una métrica de este tipo que es muy utilizada es la distancia estadística general de **Mahalanobis** que se induce a partir de las distribuciones gaussianas multivariantes. La distancia de Mahalanobis se define como:

$$D_\Sigma(\mathbf{X}, \mathbf{Y}) = \sqrt{(\mathbf{X} - \mathbf{Y})^T \Sigma^{-1} (\mathbf{X} - \mathbf{Y})} \quad (23.50)$$

donde Σ^{-1} es la matriz inversa de la de covarianzas de las muestras. Esta distancia tiene diversas propiedades interesantes. La primera es que es invariante frente a

traslaciones, rotaciones y cambios de escala de los atributos o características. Esta propiedad es particularmente interesante para evitar las distorsiones que puedan producir este tipo de transformaciones en la medición de la distancia entre muestras. La segunda propiedad es que la distancia es adimensional y relativa a la medida de dispersión de los datos. Esto es particularmente importante al considerar espacios obtenidos mediante la agregación de magnitudes de naturaleza heterogéneas. Por ejemplo, podemos considerar un conjuntos de medidas referidas a personas como puede ser la edad, peso y estatura; en cuyo caso las unidades de medidas vendrán dadas en años, kg y cm. La distancia de Mahalanobis tiene la ventaja de que escala cada medida a la dispersión de la misma, con los que las unidades de distancia resultan homogeneizadas.

23.3.2.1 Clasificador según el vecino más próximo

Sea un conjunto de c clases $\{\Omega_1, \dots, \Omega_c\}$, donde cada una de las clases Ω_i esté representada por un vector de características \mathbf{Z}_i , que denominaremos vector prototipo, o prototipo a secas, de la clase. Sea a su vez un vector de una forma incógnita \mathbf{X} , que pretendemos clasificar. La clasificación de \mathbf{X} según la regla de la mínima distancia a los prototipos se puede expresar como:

$$\mathbf{X} \in \Omega_i \leftrightarrow D(\mathbf{X}, \mathbf{Z}_i) = \min_{j=1}^c D(\mathbf{X}, \mathbf{Z}_j) \quad (23.51)$$

donde D representa al funcional de distancia definido para el espacio de representación. La fase de aprendizaje de un sistema con clasificador según la regla de decisión de la distancia mínima consistirá en obtener, a partir de las muestras de aprendizaje, los c prototipos \mathbf{Z}_i que representen a las clases correspondientes. Un vector prototipo muy utilizado es el centroide o vector medio de la clase. La clasificación por regla de distancia mínima es un caso de clasificación que puede ser reducido a una función discriminante lineal. Así, sea el caso de métrica Euclídea:

$$D^2(\mathbf{X}, \mathbf{Z}_i) = \|\mathbf{X}\|^2 + \|\mathbf{Z}_i\|^2 - 2\mathbf{Z}_i^T \mathbf{X} \quad (23.52)$$

Si definimos el siguiente clasificador lineal: $d_i(\mathbf{X}) = -\|\mathbf{Z}_i\|^2 + 2\mathbf{Z}_i^T \mathbf{X}$ donde definimos los términos: $\omega_{0i} = -\|\mathbf{Z}_i\|^2$ y $\omega_i = 2\mathbf{Z}_i$, se tiene que la mínima distancia implica el máximo de la ecuación lineal, dado que la norma $\|\mathbf{X}\|^2$ es la misma para todas las clases y por tanto es un factor que no influye en la toma de decisión entre las diferentes clases.

Las superficies fronteras entre clases son hiperplanos perpendiculares a los segmentos que unen los puntos del espacio de características que representan a los prototipos correspondientes. Además, dichos hiperplanos bisectan dicho segmento en su punto medio. Al conjunto de regiones definidas por las fronteras asociadas al clasificador de mínima distancia se las denomina **regiones de Voronoi** que conecta los problemas de clasificadores geométricos con una especialidad científica como la **Geometría Computacional**.

El clasificador de distancia mínima puede generalizarse para permitir más de un prototipo por clase. Un caso extremo es cuando cada muestra del conjunto de aprendizaje se considera un prototipo de su clase, en cuyo caso el clasificador de mínima

distancia al prototipo se denomina clasificador del **vecino más próximo** (Nearest Neighbor, NN). Este clasificador presenta diversas ventajas a pesar de ser bastante costoso computacionalmente. Las principales de ellas son su simplicidad y su generalidad. Además, no es un clasificador paramétrico en el sentido de que cada clase no debe modelarse en base al conjunto de muestras, sino que el conjunto de muestra es el modelo mismo de la clase. Es ciertamente costoso porque necesita determinar la distancia de una forma incógnita a todos las muestras de la clase, el síndrome *de la distancia a todos* que generalmente se trata de evitar en el diseño de clasificadores.

Dada la extensión de este libro, no podemos presentarlas, pero utilizando técnicas basadas en **Filtros de Condensación y Edición** de conjunto de datos podemos reducir considerablemente la complejidad de este clasificador. Existen además técnicas basadas en el Diagrama de Voronoi que permiten obtener clasificadores NN sin tener que computar la distancia a todas las muestras.

Para ilustrar el algoritmo del clasificador NN, presentaremos una codificación de clasificador NN con métrica L_2 que puede ser implementada de una forma minimalista en Matlab. Sea **Xtrain** una matriz $m \times n$ donde m es el número de muestras de un conjunto de datos de entrenamiento y n es la dimensionalidad del espacio, sea **Xtest** un vector fila, dimensión $1 \times n$, que constituye una muestra de test. Si **Ctrain** es un vector columna de m elementos que contiene la clase de cada muestra de entrenamiento, y el resultado de la clasificación es **Ctest**, la clase asignada al punto test, así como el índice **nn** del vecino más próximo, el código Matlab completo del clasificador NN es tan sencillo como:

```
function [Ctest,nn]=ClasificadorNN(Xtrain,Ctrain,Xtest)
    [Dmin,nn] = min(sum((Xtrain-repmat(Xtest,size(Xtrain,1),1)).^2,2));
    Ctest = Ctrain(nn);
end
```

El clasificador NN presenta ventajas a priori, principalmente porque permite definir reglas de clasificación sin esquemas de aprendizaje. Sin embargo, esta estructura de clasificador es de naturaleza exhaustiva y, por tanto, si el número de muestras de aprendizaje es elevado, el coste computacional en la toma de decisión también lo es.

23.3.2.2 Clasificador de los k vecinos más próximos

La clasificación según la regla del vecino más próximo puede modificarse en el sentido de que la regla no suministre el prototipo más cercano a la muestra incógnita, sino el conjunto de prototipos más cercanos. Este tipo de regla es la denominada regla de los k -vecinos más próximos (k NN), que suministra los k prototipos más próximos y a continuación, según un criterio de mayoría entre los k resultados es posible obtener la clasificación de la muestra incógnita. En un sentido probabilístico, este procedimiento puede entenderse como un clasificador Bayesiano, o pseudo-Bayesiano, por cuanto las distribución de clases entre los k vecinos más próximos constituye una estima de las distribuciones de probabilidad.

Esta regla es útil en ciertas situaciones en que las muestras de clases diferentes se encuentran muy próximas. La regla NN suministra resultados más fiables que la

kNN sólo si las distancias entre muestras de la misma clase son más pequeñas que las distancias entre muestras de diferentes clases. La principal ventaja que aporta el clasificador kNN es que la decisión se realiza en función de la mayoría de puntos que rodean al punto incógnita, lo que permite un comportamiento más suave, pero al mismo tiempo se puede obtener una mayor tasa de error. Podemos ilustrar el algoritmo kNN en Matlab como:

```
function [Ctest,Prob]=ClasificadorKNN(Xtrain,Ctrain,k,Xtest)
    [ord,index]=sort(sum((Xtrain-repmat(Xtest,size(Xtrain,1),1)).^2,2));
    Prob = hist(Ctrain(index(1:k)),size(unique(Ctrain),1))/k;
    [Pmax,Ctest] = max(Prob);
end
```

donde además de la clase resultado `Ctest` se proporciona la distribución de probabilidad `Prob` de asignación del punto de test a las distintas clases en función de los k vecinos, distribución que es una estima de $p(\Omega_i/\mathbf{X})$. Finalmente, es preciso hacer notar que la elección del número de vecinos k no puede determinarse automáticamente, sino que más bien debe ajustarse para cada aplicación hasta un valor óptimo en base a una serie de test de error con diversos valores. Aunque es evidente, recuerda que $k = 1$ genera el clasificador NN.

23.3.3 Separabilidad Lineal y la dimensión Vapnik-Chervonenkis

Un ejemplo muy sencillo de conjuntos de datos con dos clases es el conjunto de las funciones booleanas. En este caso los atributos o características son igualmente booleanas: $x_i \in \{0, 1\}$, donde se puede considerar a cualquier función booleana como un conjunto de datos con dos clases. Puede observarse fácilmente, como ilustra la Figura 23.5, que las funciones booleanas bidimensionales básicas And y Or son conjuntos de muestras linealmente separables. No es el caso de la función Or-Exclusive y su complementaria Nor-Exclusive, que sí son separables pero por discriminantes cuadráticos.

Estos ejemplos nos llevan a introducir el concepto de **Separabilidad Lineal** de un conjunto de muestras, o a su relacionado de capacidad de separación de un discriminante lineal. Matemáticamente podríamos probar que en el espacio de dimensión 2 un discriminante lineal puede separar tres puntos no colineales arbitrarios, pero no así cuatro. En el espacio de dimensión 3 podríamos separar 4 puntos no colineales arbitrarios. En general un clasificador lineal en un espacio n -dimensional puede separar $n + 1$ puntos no colineales arbitrarios.

La argumentación precedente sobre la separabilidad lineal nos lleva a presentar un concepto relacionado de carácter general sobre la capacidad discriminante de un clasificador. El mismo es la denominada **dimensionalidad Vapnik-Chervonenkis (VC) de un clasificador**. Sea $x \in \mathbf{R}^n$ las coordenadas en el espacio n -dimensional de un conjunto de datos binario y sea $y \in \{-1, +1\}$ las clases a las que pertenecen las muestras. Sea $f(x, \alpha) \in [-1, +1]$ la salida que proporciona un clasificador para

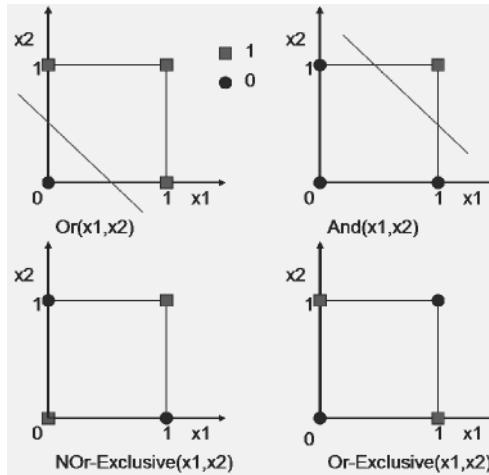


Figura 23.5: Funciones booleanas linealmente separables And y Or y las no linealmente separables Or-Exclusive y su complementaria.

el conjunto de datos, para un conjunto α de parámetros que modelan el clasificador. Se denomina pérdida del clasificador, $|y - f(x, \alpha)|$, a la diferencia en valor absoluto entre el valor de la clase a la que realmente pertenece un punto y valor proporcionado por el clasificador. La pérdida promedio se obtendrá en función de la distribución de probabilidad, $p(x)$, sobre todo el espacio de representación:

$$R(\alpha) = \int \frac{1}{2} |y - f(x, \alpha)| p(x) dx \quad (23.53)$$

Sin embargo, la estimación experimental o empírica de esta pérdida se puede aproximar sobre un conjunto finito de m muestras como:

$$R_{\text{emp}}(\alpha) = \frac{1}{2m} \sum_{i=1}^m |y_i - f(x_i, \alpha)| \quad (23.54)$$

Si $\eta \in [0, 1]$, entonces se verifica con probabilidad $1 - \eta$ la siguiente relación entre ambos valores, el teórico y el empírico:

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h(\log(2m/h) + 1) - \log(\eta/4)}{m}} \quad (23.55)$$

donde h es la dimensionalidad Vapnik-Chervonenkis (VC) del clasificador [Vapnik, 2000]. La dimensión VC es el máximo número de puntos no colineales de un conjunto de datos arbitrario que ese clasificador puede separar. Hemos visto que un clasificador lineal puede separar $n + 1$ puntos arbitrarios. Un clasificador más potente en principio sería el NN, dado que puede separar con seguridad (cuestión de costes aparte) cualquier conjunto de datos, sea cual sea su dimensionalidad y/o cardinalidad, luego su dimensión VC es potencialmente infinita.

23.4 Aprendizaje de clasificadores lineales

La tarea de aprendizaje supervisado en Reconocimiento de Formas consiste en la determinación de los parámetros del clasificador a partir de conjuntos de muestras representativas. El aprendizaje supervisado es una modalidad de aprendizaje en la que la información aportada por el usuario, o supervisión, es de vital importancia en el desarrollo del procedimiento. En la práctica, tal supervisión suele reducirse a incorporar información de la clase a la que pertenece cada muestra.

Los clasificadores lineales constituyen la forma más simple de clasificadores, en concreto el modelo biclásico es el caso más simple de todos. Se ha desarrollado una teoría de aprendizaje lineal biclásico bastante sólida, algunos de cuyos elementos expondremos a lo largo de esta sección. Representamos el conjunto de datos que se utiliza para el aprendizaje como un conjunto m muestras constituidas por pares que incluyen las coordenadas $\mathbf{X}_i \in \mathbf{R}^n$ y además la información de supervisión dada por la pertenencia a la clase como: $y_i \in \{-1, 1\}$. Teniendo en cuenta esta notación, un clasificador lineal con coeficientes w_0 y \mathbf{w} desconocidos debe verificar la Expresión 23.42, que transformada se puede expresar de la siguiente forma:

$$y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) > 0 \quad (23.56)$$

Si representamos por \mathbf{Y}_i^T a las coordenadas homogéneas y por \mathbf{Z}_i^T a las homogéneas y además normalizadas por el vector de clase y_i :

$$\mathbf{Z}_i^T = y_i \mathbf{Y}_i^T = y_i(1 \ \mathbf{X}_i^T) = (\begin{array}{cccc} y_i & y_i x_{i1} & \cdots & y_i x_{in} \end{array}) \quad (23.57)$$

La expresión que ha de verificar cada muestra es la siguiente: $y_i \mathbf{Y}_i^T \mathbf{a} = \mathbf{Z}_i^T \mathbf{a} > 0$. Si existe un total de m muestras de aprendizaje, el sistema total de ecuaciones del problema de aprendizaje está dado por:

$$\Psi \mathbf{a} > \mathbf{0} \quad (23.58)$$

donde $\Psi \mathbf{a}$ es un vector de dimensión $m \times 1$, y cada componente de este vector es $\mathbf{Z}_i^T \mathbf{a}$. La matriz Ψ de dimensión $m \times (n + 1)$ está dada por las coordenadas homogéneas normalizadas por la clase:

$$\Psi = \left(\begin{array}{c} \mathbf{Z}_1^T \\ \vdots \\ \mathbf{Z}_m^T \end{array} \right) = \left(\begin{array}{cccc} y_1 & y_1 x_{11} & \cdots & y_1 x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ y_m & y_m x_{m1} & \cdots & y_m x_{mn} \end{array} \right) \quad (23.59)$$

La Expresión 23.58 define el problema de aprendizaje de clasificadores lineales, donde la matriz Ψ es conocida y se pretende encontrar el vector homogéneo \mathbf{a} que define el clasificador. La primera observación que hay que realizar es la referente a la existencia de soluciones. A partir de la experiencia práctica de este tipo de problemas podemos considerar dos tipos de situaciones. La primera es que no exista solución, porque el conjunto de muestras no sea separable linealmente. La segunda situación es que exista solución, pero en tal caso generalmente no suele ser única, sino que existe

toda una familia de soluciones posibles. Fácilmente se puede comprobar que la existencia de una solución implica la existencia de infinitas por la simple transformación: $\mathbf{a} \rightarrow \lambda\mathbf{a}$, con $\lambda > 0$ que igualmente verifica la referida expresión. Pero este caso es el más simple, dado que ambas soluciones son proporcionales. Existen sin embargo soluciones múltiples cuya relación no es tan simple. La Figura 23.6 ilustra un ejemplo de conjunto de datos con dos clases linealmente separables que admite múltiples soluciones del clasificador lineal. El establecimiento del problema formal de aprendizaje es sencillo, pero la solución no es fácil. Los distintos métodos existentes de aprendizaje de clasificadores lineales difieren en la estrategia utilizada para intentar resolver la Ecuación 23.58.

23.4.1 Procedimiento perceptrón

A finales de los años cincuenta y principios de los sesenta, se desarrollaron una clase de máquinas diseñadas por Rosemblatt denominadas habitualmente como Perceptrones, que parecieron ofrecer a muchos investigadores un modelo natural y potente de máquina de aprendizaje. Aunque hoy día se considera que las expectativas que se crearon en lo referente a las prestaciones del Perceptrón eran excesivamente optimistas, los conceptos matemáticos que surgieron de su desarrollo continúan jugando un papel de cierta relevancia en la teoría de Reconocimiento de Formas, además de haber propiciado el advenimiento de sistemas más complejos como las Redes Neuronales.

Actualmente, el Perceptrón designa una familia de procedimientos que presentan algunas variantes entre ellos. No presentaremos el Perceptrón original de Rosemblatt, sino una versión posterior más sencilla formalmente. Expondremos en primer lugar el caso biclásico más sencillo. Sea un vector \mathbf{V} y denominemos por $\|\mathbf{V}\|$ a su norma y por $|\mathbf{V}|$ a un vector cuyas componentes son el valor absoluto de las de \mathbf{V} . Sea que para la muestra i de un conjunto de datos construimos la siguiente función dependiente de la expresión $\mathbf{Z}_i^T \mathbf{a}$:

$$J_i(\mathbf{a}) = \frac{1}{2}(|\mathbf{Z}_i^T \mathbf{a}| - \mathbf{Z}_i^T \mathbf{a}) = \begin{cases} 0 & \mathbf{Z}_i^T \mathbf{a} > 0 \\ -\mathbf{Z}_i^T \mathbf{a} & \mathbf{Z}_i^T \mathbf{a} < 0 \end{cases} \quad (23.60)$$

Recordando que se verifica $\mathbf{Z}_i^T \mathbf{a} > 0$ cuando una muestra se encuentra correctamente clasificada, la función $J_i(\mathbf{a})$ es siempre no negativa siendo nula cuando la muestra se encuentra correctamente clasificada y positiva cuando se encuentra mal clasificada. Podemos considerar a $J_i(\mathbf{a})$ como una medida de error respecto a la correcta clasificación de la muestra i . Una medida extendida a todo el conjunto de datos sería:

$$J(\mathbf{a}) = \sum_{i=1}^m J_i(\mathbf{a}) \quad (23.61)$$

Esta función tiene un valor nulo cuando los parámetros del clasificador son tales que todas las muestras se encuentran correctamente clasificadas y un valor positivo en caso contrario. Es decir, podemos establecer que para encontrar un clasificador lineal correcto deberíamos encontrar los argumentos \mathbf{a} que hacen mínima $J(\mathbf{a})$. Esta

es la estrategia seguida por el procedimiento Perceptrón para encontrar la solución a la Ecuación (23.58). En la práctica este procedimiento utiliza el método del gradiente para alcanzar la minimización de la función objetivo. El gradiente está dado por:

$$\nabla_{\mathbf{a}} J(\mathbf{a}) = - \sum_{\mathbf{Z}_i^T \mathbf{a} < 0} \mathbf{Z}_i \quad (23.62)$$

donde la sumatoria se extiende sobre todas las muestras que incumplen la condición de clasificación correcta. El procedimiento utiliza un método de aproximaciones sucesivas que partiendo de un valor arbitrario inicial $\mathbf{a}^{(0)}$, realiza una corrección sobre los valores actuales de los parámetros del clasificador lineal:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \rho^{(k)} \nabla_{\mathbf{a}} J(\mathbf{a}^{(k)}) = \mathbf{a}^{(k)} + \rho^{(k)} \sum_{\mathbf{Z}_i^T \mathbf{a}^{(k)} < 0} \mathbf{Z}_i \quad (23.63)$$

hasta que se verifique algún criterio de convergencia, por ejemplo: $\|\mathbf{a}^{(k+1)} - \mathbf{a}^{(k)}\| < \epsilon$. Además, $\rho^{(k)}$ es un valor de proporcionalidad que puede ser constante o dependiente de la iteración en base a diversas estrategias de optimización. El procedimiento converge a una solución en un número finito de iteraciones si se cumplen ciertas condiciones sobre $\rho^{(k)}$ [Duda y otros, 2001, pag. 233]. La expresión anterior define una variante del procedimiento Perceptrón de corrección basada en múltiples muestras o **Batch Perceptrón**, porque en la corrección de los coeficientes se hacen intervenir todas las muestras del conjunto de datos incorrectamente clasificadas. Existen otras variantes que hacen intervenir secuencialmente sólo una muestra en cada iteración.

La idea de extender el Perceptrón, que es un procedimiento intrínsecamente binomial, a problemas con múltiples clases ha producido algunas generalizaciones del mismo. La primera dificultad que surge es que el procedimiento en sí mismo no es generalizable, por cuanto su esencia reside en la existencia de una única frontera de decisión lineal, que no existe en el problema con múltiples clases. Para superar esta dificultad, se han propuesto diversas soluciones. La más sencilla consiste en construir una serie de perceptrones tal que cada uno separa las muestras de cada clase de las muestras de las restantes. En este caso, se puede realizar diversas ejecuciones del procedimiento, tantas como clases, definiendo en cada caso como muestras positivas la de la clase en cuestión y como negativas las muestras pertenecientes a las demás clases. La principal crítica que se puede hacer a este procedimiento es que no genera un clasificador multiclásico, sino más bien c clasificadores binarios [Duda y otros, 2001, sec. 5.2.2]. La presentación del procedimiento puede simplificarse introduciendo una matriz y_{ij} similar a y_i pero ampliado a c clases en la forma siguiente:

$$y_{ij} = \begin{cases} 1 & \mathbf{X}_i \in \Omega_j \\ -1 & \text{en caso contrario} \end{cases} \quad (23.64)$$

en cuyo caso podemos obtener los c clasificadores simultáneamente con la ecuación de correcciones de los coeficientes de los clasificadores lineales de la forma siguiente:

$$\begin{aligned} \mathbf{a}_j^{(k+1)} &= \mathbf{a}_j^{(k)} + \rho^{(k)} y_{ij} \mathbf{Y}_i & y_{ij} \mathbf{Y}_i^T \mathbf{a}_j^{(k)} < 0 \\ \mathbf{a}_j^{(k+1)} &= \mathbf{a}_j^{(k)} & \text{en caso contrario} \end{aligned} \quad (23.65)$$

23.4.2 Procedimientos de mínimo error cuadrático

Los procedimientos de mínimo error cuadrático se basan en utilizar funciones objetivo de naturaleza menos heurística, pero con mayor significado. Se recurre a un concepto de significado bien conocido como es el de error, que podemos considerar con carácter general como la desviación del comportamiento observado respecto al deseado. Podemos considerar dos grupos de procedimientos de error cuadrático mínimo. El primer grupo contiene aquellos procedimientos que trabajan sobre el discriminante lineal con márgenes de seguridad basados en la expresión:

$$\Psi \mathbf{a} > \mathbf{b} > 0 \quad (23.66)$$

El segundo grupo contiene aquellos que trabajan sobre la formulación del error producido por la máquina del clasificador lineal. La importancia de estos métodos reside en que son la base de los clasificadores no lineales en forma de redes de perceptrones simples. La respuesta de este sistema es: $y = f(\mathbf{Y}_i^T \mathbf{a})$. El error se obtiene como la diferencia entre el comportamiento verificado y el deseado que está definido por y_i para el caso biclásico.

$$e_i(\mathbf{a}) = y_i - f(\mathbf{Y}_i^T \mathbf{a}) \quad (23.67)$$

El primer grupo de procedimientos se basan en introducir un vector de márgenes $b_i > 0$ para cada muestra, de forma que se exige que la ecuación del clasificador lineal se verifique con un valor de al menos ese margen, es decir $\mathbf{Z}_i^T \mathbf{a} > b_i$. Según que el vector \mathbf{b} de dimensión $m \times 1$ sea un dato que el usuario aporta, o bien una incógnita en sí misma, se tendrán diversas variantes que estudiaremos seguidamente. En el primer caso se trabaja con el error $\mathbf{e} = \Psi \mathbf{a} - \mathbf{b}$, en la forma:

$$J(\mathbf{a}) = \|\mathbf{e}\|^2 = \sum_{i=1}^m (\mathbf{Z}_i^T \mathbf{a} - b_i)^2 = \|\Psi \mathbf{a} - \mathbf{b}\|^2 \quad (23.68)$$

La condición que ha de verificar la solución es: $\nabla_{\mathbf{a}} J(\mathbf{a}) = 0$, que se expresa de la forma siguiente:

$$\nabla_{\mathbf{a}} J(\mathbf{a}) = 2 \sum_{i=1}^m \mathbf{Z}_i (\mathbf{Z}_i^T \mathbf{a} - b_i) = 2 \Psi^T (\Psi \mathbf{a} - \mathbf{b}) = 0 \quad (23.69)$$

La matriz Ψ tiene dimensiones de $m \times (n + 1)$, por tanto Ψ^T tiene dimensión $(n + 1) \times m$ y por tanto la matriz producto $\Psi^T \Psi$ tiene dimensiones $(n + 1) \times (n + 1)$ siendo pues una matriz cuadrada. La solución a la ecuación anterior es:

$$\mathbf{a} = (\Psi^T \Psi)^{-1} \Psi^T \mathbf{b} = \Psi^{\dagger} \mathbf{b} \quad (23.70)$$

donde Ψ^{\dagger} es la matriz pseudoinversa de Ψ , que verifica: $\Psi^{\dagger} \Psi = \mathbf{I}$ y $\Psi \Psi^{\dagger} \neq \mathbf{I}$.

23.4.2.1 Procedimiento Widrow-Hoff

También denominado procedimiento **LMS** (least-mean-square) o procedimiento **Adaline** [Theodoridis y Koutroumbas, 2003, pag. 70]. En este caso, se fija el vector \mathbf{b} y se computan las correcciones en base al gradiente definido anteriormente. Se tiene:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \rho^{(k)} \nabla_{\mathbf{a}} J(\mathbf{a}^{(k)}) = \mathbf{a}^{(k)} + \rho^{(k)} \Psi^T (\mathbf{b} - \Psi \mathbf{a}^{(k)}) \quad (23.71)$$

La principal ventaja es que no precisa computar la matriz pseudo-inversa y los problemas que pudiera implicar su singularidad. La principal dificultad de este procedimiento es que precisa definir como datos los márgenes para cada muestra, lo que influye en la decisión final. Desgraciadamente, este procedimiento no siempre converge a una solución que separe las clases aun cuando la solución exista, o mejor dicho converge a una solución que no es la correcta [Duda y otros, 2001, pag. 247].

23.4.2.2 Procedimiento Ho-Kashyap

El procedimiento Ho-Kasyap considera que los márgenes \mathbf{b} son incógnitas que el procedimiento debe a su vez determinar. Realmente se tiene una función objetivo con dos tipos de variables:

$$J(\mathbf{a}, \mathbf{b}) = \|\Psi \mathbf{a} - \mathbf{b}\|^2 \quad (23.72)$$

En tal caso se debe tener dos ecuaciones del gradiente. Por un lado: $\nabla_{\mathbf{a}} J(\mathbf{a}, \mathbf{b})$ y por otro lado: $\nabla_{\mathbf{b}} J(\mathbf{a}, \mathbf{b})$. La primera ya la hemos considerado y conduce a que: $\mathbf{a} = \Psi^{\dagger} \mathbf{b}$, que nos permite computar la variable \mathbf{a} a partir de la \mathbf{b} , que queda como única variable del problema. La segunda expresión nos permite realizar correcciones de los márgenes; pero como estos deben ser variables positivas el Procedimiento Ho-Kasyap propone una modificación de la regla del gradiente en el siguiente sentido:

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} - \rho^{(k)} \frac{1}{2} (\nabla_{\mathbf{b}} J - |\nabla_{\mathbf{b}} J|) \quad (23.73)$$

de tal forma que las modificaciones del vector de margen nunca lo hacen negativo. Después de algunos cálculos se llega a la siguiente expresión:

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + 2\rho^{(k)} \mathbf{e}^{(k)+} \quad (23.74)$$

donde $\mathbf{e}^{(k)+}$ es la parte positiva del error $\mathbf{e}^{(k)} = \Psi \mathbf{a}^{(k)} - \mathbf{b}^{(k)}$ dado por:

$$\mathbf{e}^{(k)+} = \frac{1}{2} (\mathbf{e}^{(k)} + |\mathbf{e}^{(k)}|) \quad (23.75)$$

El procedimiento Ho-Kasyap parte de la computación previa de la matriz pseudo-inversa y de una elección arbitraria y positiva del vector de márgenes: $\mathbf{b} > 0$ y de \mathbf{a} dependiente de las anteriores variables. Una de las grandes ventajas del procedimiento Ho-Kasyap es que permite detectar en la ejecución del procedimiento cuando no existe solución. En los anteriores procedimientos no sabemos tal hecho, dado que finalizarán sin convergencia solo por haber agotado un límite superior de iteraciones. Si cuando el

procedimiento Ho-Kasyap ha finalizado se detecta que el error es negativo en alguna muestra, entonces no se verifica: $\mathbf{e} = \Psi\mathbf{a} - \mathbf{b} \geq 0$ y por tanto no existe solución del problema. Pero ello sólo ocurre cuando el conjunto de datos no es separable linealmente. Podemos sistematizar los finales diferentes de este procedimiento en la forma siguiente basándonos en el vector de errores y de partes positivas de los errores.

1. Si alguna componente del vector de error es positiva, no se ha finalizado el procedimiento.
2. Si todos los errores son no positivos, es decir, la parte positiva de los errores es nula, se ha finalizado el procedimiento.
 - Si todos los errores son nulos, se ha encontrado una solución.
 - Si algún error es negativo, no existe solución.

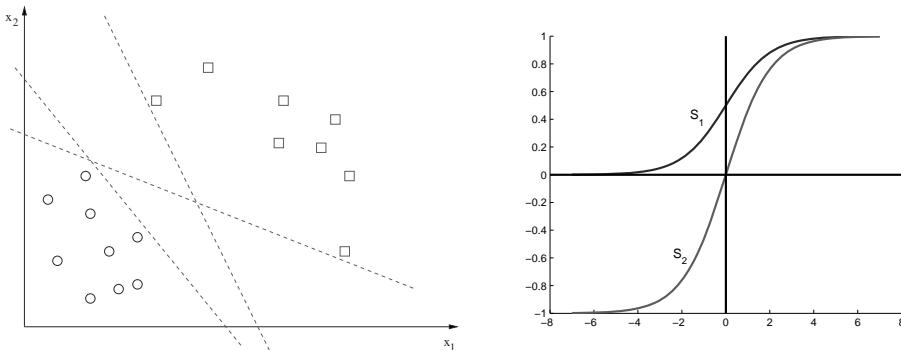


Figura 23.6: A la izquierda, ilustración de la multitud de soluciones para el clasificador lineal. A la derecha, funciones sigmoide que suavizan la función de decisión del tipo todo o nada, haciéndolas derivables, $S_1 \in [0, 1]$ y $S_2 \in [-1, +1]$. En la representación gráfica es: $\beta = 1$

23.4.2.3 Máquina lineal de mínimo error

En este caso pretendemos construir una máquina lineal que presente un error mínimo entre la respuesta que produce y la que debiera producir. El planteamiento es distinto al estudiado a lo largo de este tema, pues en este caso no se trabaja sobre la ecuación de desigualdad del discriminante. Se trabaja con la respuesta de una máquina lineal, que en su momento definimos como:

$$o_i(\mathbf{a}) = f(\mathbf{Y}_i^T \mathbf{a}) \quad (23.76)$$

donde $f()$ es una función de activación. Para evitar problemas de derivabilidad en el cálculo del gradiente, se utiliza una función suavizada de tipo *sigmoide*. Podemos definir dos tipos diferentes de estas funciones:

$$S_1(x, \beta) = \frac{1}{1 + e^{-\beta x}} \in [0, 1] \quad S_2(x, \beta) = \frac{1 - e^{-\beta x}}{1 + e^{-\beta x}} \in [-1, +1] \quad (23.77)$$

donde β es un parámetro y cuyas derivadas son:

$$\frac{dS_1}{dx} = \beta(1 - S_1)S_1 \quad \frac{dS_2}{dx} = \frac{1}{2}\beta(1 - S_2^2) \quad (23.78)$$

Estas funciones tienen la ventaja de que el valor de la derivada puede calcularse directamente del valor de la función. La Figura 23.6 representa las curvas de ambas funciones. La función objetivo que se pretende minimizar es la norma de la diferencia o error entre la respuesta proporcionada por la máquina lineal y la que debiera producir para el conjunto de muestra de entrenamiento, es decir:

$$J(\mathbf{a}) = \sum_{i=1}^m (y_i - o_i(\mathbf{a}))^2 = \sum_{i=1}^m (y_i - f(\mathbf{Y}_i^T \mathbf{a}))^2 \quad (23.79)$$

Denominaremos al error como: $e_i(\mathbf{a}) = y_i - f(\mathbf{Y}_i^T \mathbf{a})$. El gradiente para el primer tipo de sigmoide lo podemos computar como:

$$\nabla_{\mathbf{a}} J(\mathbf{a}) = -\beta \sum_{i=1}^m e_i(\mathbf{a}) (1 - o_i(\mathbf{a})) o_i(\mathbf{a}) \mathbf{Y}_i \quad (23.80)$$

Las actualizaciones de los coeficientes viene dada por:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \rho^{(k)} \beta \sum_{i=1}^m e_i(\mathbf{a}) (1 - o_i(\mathbf{a})) o_i(\mathbf{a}) \mathbf{Y}_i \quad (23.81)$$

Obsérvese que las correcciones son directamente proporcionales al error detectado. Este hecho será utilizado posteriormente cuando generalicemos las máquinas lineales. Dado que el valor del vector de coeficientes puede encontrarse en cualquier rango, se pueden normalizar multiplicando por un factor, dado que la transformación $\mathbf{a} \rightarrow \lambda \mathbf{a}$ no afecta al resultado. Una posible normalización consiste en exigir que el vector \mathbf{w} del resultado sea un vector unitario. Esto se consigue introduciendo la siguiente normalización:

$$\mathbf{a} \leftarrow \frac{1}{\|\mathbf{w}\|} \mathbf{a} = \frac{1}{\|\mathbf{w}\|} \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix} \quad (23.82)$$

23.4.3 Máquina de vectores soportes

Todos los procedimientos presentados para obtener clasificadores tratan de proporcionar una solución válida de entre las muchas posibles. En algunos casos, de forma no infrecuente, la frontera de separación entre clases se encuentra muy cerca de alguna de las muestras fronterizas y simultáneamente muy alejada de las demás, véase la Figura 23.6. En estos casos los márgenes de separación de las muestras al clasificador

lineal son muy desiguales. Dado que en un sistema de Reconocimiento de Formas es frecuente la existencia de ruido, este puede producir un desplazamiento de alguna muestra y causar error, que es más probable si existe un fuerte desequilibrio de los márgenes. Un objetivo adicional al de obtener un clasificador válido, es el de obtener un clasificador con el mayor margen de separación posible para minimizar el efecto del ruido en el sistema de Reconocimiento de Formas.

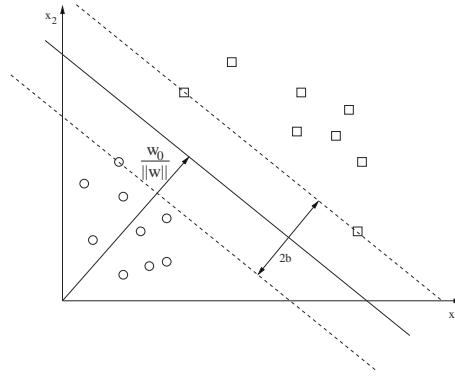


Figura 23.7: Geometría del problema de la Máquina de Vectores Soporte.

El procedimiento de la Máquina de Vectores Soportes (SVM, Support Vector Machine) [Cristiani y Taylor, 2000] trata de determinar un clasificador que tenga una banda lo más ancha posible entre las muestras más próximas de ambas clases. El clasificador óptimo será aquel que atraviesa esa banda por la mitad, tal y como ilustra la Figura 23.7. La restricciones que han de verificar las muestras en base al clasificador serán en la forma siguiente:

$$y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) > b \quad (23.83)$$

donde b es el margen desconocido que implica la existencia de una banda de separación de $2b$ entre las muestras de ambas clases. Nótese que existe una dirección del clasificador, y sólo una, para la cual esta banda es máxima. La mínima distancia del clasificador al origen está dada por:

$$D = \frac{w_0}{\|\mathbf{w}\|} \quad (23.84)$$

mientras que las líneas que delimitan las bandas están situadas a distancias $(w_0 + b)/\|\mathbf{w}\|$ y $(w_0 - b)/\|\mathbf{w}\|$ respectivamente. El ancho de la banda medido perpendicularmente al origen estará dado por:

$$\Delta = \frac{w_0 + b}{\|\mathbf{w}\|} - \frac{w_0 - b}{\|\mathbf{w}\|} = \frac{2b}{\|\mathbf{w}\|} \quad (23.85)$$

Por tanto, el objetivo a alcanzar es el de un clasificador que maximice Δ , o minimice $1/\Delta$, y cumpla con las restricciones de la Expresión 23.83. Pero en esta expresión

aparecen de forma proporcional tres variables independientes b , w_0 y \mathbf{w} , que pueden ser sometidas a una normalización de amplitud. El criterio que se utiliza consiste en normalizar respecto al valor de b , de tal forma que las restricciones se convierten en valores normalizados en:

$$y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) > 1 \quad (23.86)$$

En cuyo caso el objetivo sería maximizar $2/\|\mathbf{w}\|$ o bien minimizar $\|\mathbf{w}\|^2/2$. Se elige esta última opción por cuanto el cuadrado de la norma es una expresión cuadrática que constituye una alternativa computacionalmente más atractiva que la otra opción. Podemos resumir el problema en las siguientes ecuaciones:

$$\min \frac{1}{2} \|\mathbf{w}\|^2 = \min \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) > 1 \quad (23.87)$$

Este problema es cuadrático con restricciones lineales de desigualdad, por ello resulta relativamente sencillo de resolver mediante técnicas de programación cuadrática. Puesto en formato homogéneo resulta:

$$\min \frac{1}{2} \mathbf{a}^T \mathbf{Q} \mathbf{a} \quad \mathbf{\Psi} \mathbf{a} > 1 \quad (23.88)$$

que puede ponerse en forma de desigualdad canónica: $-\mathbf{\Psi} \mathbf{a} < -1$ y donde \mathbf{Q} es una matriz casi-identidad, es decir $Q_{11} = 0$, $Q_{ii} = 1$, y $Q_{ij} = 0, i \neq j$. Una codificación minimalista en Matlab de esta versión de SVM que denominaremos primal es la siguiente que obtiene los multiplicadores de Lagrange, cuya utilidad explicaremos posteriormente, y que se basa en una función de optimización cuadrática [MAT, 2006]:

```
function [a,lambda,exitflag] = svm_primal(Psi)
    [m,n] = size(Psi);
    Q = eye(n);Q(1,1)=0;
    [a,fval,exitflag,out,lam] = quadprog(Q,[],-Psi,-ones(m,1));
    lambda = lam.ineqlin;
end
```

Otra forma de resolver el problema es transformarlo en su problema dual, para ello intentemos resolver el problema analíticamente. Debemos aplicar la minimización de Lagrange acompañada de las condiciones de Kuhn-Tucker [Winston, 2005] para problemas de optimización con restricciones. La función de Lagrange es:

$$L(\mathbf{w}, w_0) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^m \lambda_i [y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) - 1] \quad (23.89)$$

Con los multiplicadores de Lagrange positivos, $\lambda_i \geq 0$, asociados cada uno a una muestra. Las condiciones adicionales de Kuhn-Tucker son:

$$\lambda_i [y_i(w_0 + \mathbf{X}_i^T \mathbf{w}) - 1] = 0 \quad (23.90)$$

Estas condiciones establecen que cuando $\lambda_i \neq 0$, entonces debe ser $y_i(w_0 + \mathbf{X}^T \mathbf{w}) - 1 = 0$ y por tanto las muestras correspondientes se encuentran en la frontera de las bandas. Tales muestras en la frontera de las bandas se denominan los **Vectores Soporte** del conjunto de datos. Por ello, los **multiplicadores de Lagrange** λ_i nos informan sobre la posición de las muestras respecto a los límites de la banda. Esta propiedad nos permite computar el término w_0 en la forma siguiente:

$$w_0 = \frac{1}{y_k} - \mathbf{X}_k^T \mathbf{w} \quad \lambda_k \neq 0 \quad (23.91)$$

Para proceder a resolver el problema por el segundo método, derivaremos la función de Lagrange:

$$\frac{\partial L}{\partial w_0} = - \sum_{i=1}^m \lambda_i y_i = 0 \quad (23.92)$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^m \lambda_i y_i \mathbf{X}_i = 0 \quad (23.93)$$

De la última expresión podemos computar cuánto vale el vector \mathbf{w} si conocemos los multiplicadores de Lagrange. Podemos transformar la expresión del Lagrangiano L sustituyendo estos dos valores en la Expresión 23.89, y que da lugar a:

$$L(\boldsymbol{\lambda}) = \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y_i y_j \mathbf{X}_i^T \mathbf{X}_j \quad (23.94)$$

sometida a las restricciones:

$$\sum_{i=1}^m \lambda_i y_i = 0 \quad \lambda_i \geq 0 \quad (23.95)$$

Si introducimos la matriz $H_{ij} = y_i y_j \mathbf{X}_i^T \mathbf{X}_j$, dependiente únicamente del conjunto de datos, se puede simplificar la expresiones matemáticas. Nótese que esta matriz \mathbf{H} es el producto escalar de todas las muestras entre sí, normalizados con un signo negativo cuando son de clases diferentes. Una vez calculada esta matriz el problema sólo depende de los multiplicadores de Lagrange. El problema se formula como:

$$\max_{\boldsymbol{\lambda}} \left[\sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j H_{ij} \right] \quad \sum_{i=1}^m \lambda_i y_i = 0 \quad \lambda_i \geq 0 \quad (23.96)$$

La razón por la que el problema es de maximización es porque es un problema dual del primigenio, denominado primal. Más detalles sobre las relaciones entre problemas de optimización y sus problemas duales pueden encontrarse en documentación específica relativa a materias de Investigación Operativa [Winston, 2005]. La ventaja de la forma dual de SVM es que su resolución no depende de la dimensionalidad del espacio de representación del problema, es decir es más eficiente si la dimensionalidad es grande y el número de muestras es pequeño.

23.5 Aprendizaje de clasificadores no lineales

Además de los clasificadores NN, k NN y Bayesianos, es posible definir clasificadores para clases no linealmente separables utilizando perceptrones dispuestos en múltiples capas, que son redes sin realimentaciones con una o más capas de nodos entre las entradas y las salidas. Estas capas adicionales contienen unidades ocultas, o nodos que no están conectados directamente a las entradas ni a las salidas. El perceptrón multicapa es el primer antecedente de Red Neuronal, pero en esa disciplina se estudian otras topologías más complejas que la considerada en este capítulo. En esta sección presentaremos los perceptrones multicapa como clasificadores capaces de separar cualquier conjunto de datos y su extensión en las Redes de Base Radial.

23.5.1 Topología de perceptrones multicapa

Los perceptrones tanto simples como en multicapa generan regiones de discriminación o clasificación que tienen propiedades geométricas notables, conectando con conceptos de geometría computacional tales como son las regiones convexas. En principio para realizar una reflexión general nos referiremos a perceptrones con funciones de activación de naturaleza booleana, bien sea con salidas $\{-1, +1\}$ o bien $\{0, 1\}$. Un perceptrón simple genera una estructura de clasificación definida por un hipervolumen que separa el espacio multidimensional en dos regiones abiertas, siendo ambas convexas.

Como ilustra la Figura 23.8, para comprender la topología de la decisión de un perceptrón de dos capas, es necesario referirse a un modelo simplificado del mismo en el que la segunda capa realiza una función similar a un **And** booleano. En este caso, el perceptrón de dos capas puede asimilarse a una conjunción de discriminantes lineales lo que genera una región de decisión convexa. En el marco de este mismo modelo simplificado, un perceptrón de tres capas puede asimilarse a uno en el que la tercera capa realiza una función booleana **Or**. En este caso tendremos una disjunción de regiones, pudiéndose obtener cualquier región de decisión siempre que podamos descomponerla en regiones convexas más simples. Realmente las capas segunda y tercera pueden sintetizar cualquier función booleana, lo que implica que el perceptrón simplificado puede sintetizar cualquier región de decisión que pueda formarse con tramos lineales.

Obsérvese que con un número suficientemente grande de tramos lineales asimilables a elementos de proceso de la primera capa, se puede configurar una topología de clasificación que separe convenientemente cualquier conjunto de datos. Si las funciones de activación son continuas, por ejemplo sigmoides, las fronteras de decisión formadas por tramos lineales se suavizan, transformándose en continuas. Esto ilustra que la capacidad de un perceptrón de tres capas está referida a su gran plasticidad, de manera que con suficientes elementos en las capas intermedias y un proceso de ajuste de los pesos pueden formarse sistemas de clasificación tan complejas como se deseé.

En la práctica, los perceptrones multicapa que se utilizan no son tan idealizados y la funcionalidad de las distintas capas y elementos de cómputo no están tan claras

como en la simplificación mostrada, que lo es solamente a efectos ilustrativos. En los perceptrones multicapas se utilizan funciones de activación continuas que permiten la continuidad y derivabilidad de la función de transferencia al efecto de poder generar procedimientos prácticos de entrenamiento. En una red neuronal como la utilizada en problemas de clasificación no lineal, ilustrada en la Figura 23.9, hemos de destacar dos tipos de elementos:

1. **Los elementos visibles de la red**, que se corresponden con las variables de entradas y salidas de la misma que tienen un significado concreto en el ámbito de la aplicación.
2. **Los elementos ocultos de la red**, que son todas las unidades de proceso y capas de variables internas. Estos carecen de significado traducible directamente a una explicación simbólica y son *metodológicamente* invisibles u ocultos y carentes de significados concretos.

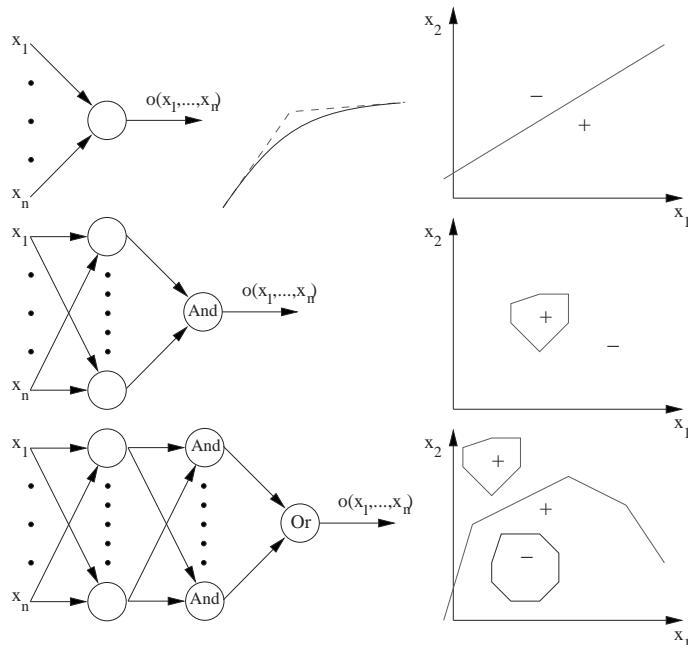


Figura 23.8: Topología de las regiones de decisión para perceptrones simplificados de una, dos y tres capas. En el primer caso la región de decisión es un semiplano, en el segundo una región convexa abierta o cerrada y en el tercer caso cualquier juxtaposición de regiones convexas, incluyendo agujeros convexas. Con funciones de activación sin discontinuidades las fronteras de las regiones se suavizan eliminando las angulosidades.

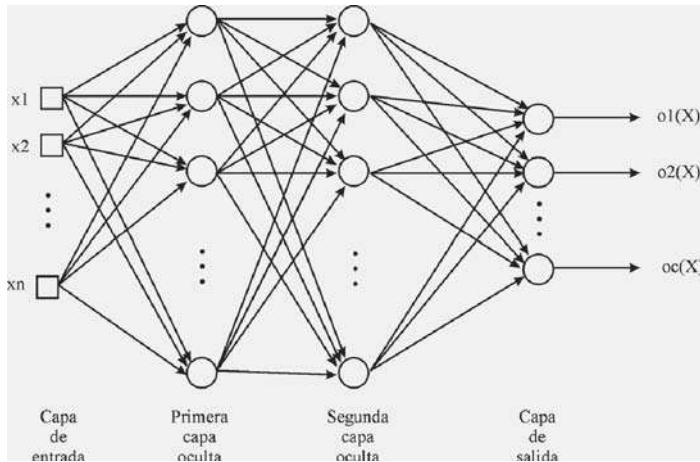


Figura 23.9: Modelo de Red Neuronal de tres capas de proceso.

El procedimiento de Retropropagación (*backpropagation*) permite entrenar redes de perceptrones multicapa mediante un ingenioso mecanismo de corrección de los pesos de la red que se inicia en la última capa y que avanza hacia la primera capa, de ahí su nombre de propagación hacia atrás. Los detalles de este procedimiento se presentan en el capítulo de este texto dedicado a Redes Neuronales.

23.5.2 Redes de funciones de base radial

Los perceptrones de tres capas proporcionan clasificadores universales que son capaces de adaptarse a un conjunto de datos arbitrario, pero su principal inconveniente lo constituye un procedimiento masivo, de fuerza bruta, con infinidad de elementos lineales y parámetros que ajustar y sin un conocimiento preciso sobre si tales elementos son o no necesarios para una aplicación dada. Las redes neuronales basadas en Funciones de Base Radial (Radial Based Functions, RBF) [Duda y otros, 2001, sec. 6.10] [Bishop, 1995, cap. 5] proporcionan una alternativa conceptualmente más simple. En este caso se reduce el número de capas mediante la introducción de una capa de transformaciones no lineales y un clasificador lineal final, tal y como se muestra en la Figura 23.10. Una exposición rigurosa de los fundamentos subyacentes puede ser presentada mediante la Teoría de Núcleos, [Cristiani y Taylor, 2000, cap. 3], pero una versión más intuitiva puede introducirse mediante el llamado Teorema de Cover [Haykin, 1998, sec. 7.2].

Una exposición más detallada del tema se encuentra en el capítulo 15. En este capítulo sólo las presentamos de una forma resumida. Realizaremos primeramente un cambio de coordenadas mediante un conjunto de transformaciones no lineales $x \rightarrow z$, de tal forma que la representación del conjunto de datos inicialmente basado en las coordenadas x se realiza en el espacio transformado de coordenadas z de mayor dimensionalidad:

$$z_i = \varphi_i(x_1, \dots, x_n) \quad i = 1, \dots, l \quad l > n \quad (23.97)$$

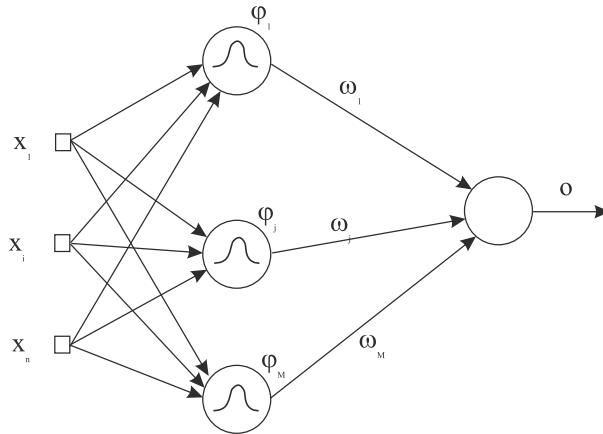


Figura 23.10: Clasificador no lineal basado en una Red de Funciones de Base Radial con M unidades de transformación no lineal.

El teorema de Cover establece que la probabilidad de separabilidad lineal en el espacio transformado z aumenta con la dimensionalidad l . Es decir, **cuanto mayor sea la dimensionalidad del espacio transformado, mayor será la probabilidad de que un problema no linealmente separable en las coordenadas x sí lo sea en las coordenadas z** . En la práctica, una forma sistemática de introducir transformaciones no lineales es mediante un conjunto de funciones radiales en la forma siguiente:

$$z_i = \varphi(\|\mathbf{x} - \mathbf{c}_i\|) \quad i = 1, \dots, l \quad (23.98)$$

donde \mathbf{c}_i es un conjunto de puntos en el espacio de x . Estas funciones se denominan radiales pues dependen únicamente del radio determinado por la norma: $\|\mathbf{x} - \mathbf{c}_i\|$. En muchas aplicaciones prácticas, después de transformar el espacio de representación mediante un conjunto de funciones del tipo especificado, se suele utilizar un clasificador SVM para realizar la separación lineal final. Denominaremos a tal solución como un SVM no lineal [Web, 2002, sec. 5.4], que resulta de la agregación de dos técnicas diferentes. En el caso más general el problema consiste en determinar no sólo los parámetros del clasificador lineal final, sino también los centros \mathbf{c}_i mismos. La respuesta de la red vendrá dada por:

$$o(\mathbf{x}_j, \boldsymbol{\omega}, \mathbf{c}) = f \left(\omega_0 + \sum_{i=1}^M \omega_i \varphi_i(\|\mathbf{x} - \mathbf{c}_i\|) \right) \quad (23.99)$$

Una familia de funciones bastante utilizadas como funciones de base radial son las gaussianas: $\varphi(u) = \exp(u)$. Una vez definida la estructura de la RBF con la utilización,

por ejemplo, de las funciones gaussianas, el conjunto de parámetros que definen su comportamiento son:

- El número de unidades en la capa oculta M .
- Los centros de estas unidades \mathbf{c}_i .
- El conjunto de pesos ω_i .

Este conjunto de parámetros han de ser obtenidos en un proceso de aprendizaje que se basa en minimizar el error para el conjunto de m muestras:

$$J(\boldsymbol{\omega}, \mathbf{c}) = \sum_{j=1}^m (y_j - o(\mathbf{x}_j, \boldsymbol{\omega}, \mathbf{c}))^2 \quad (23.100)$$

23.6 Selección de características

La complejidad de los procedimientos de aprendizaje en general, y de los clasificadores particularmente depende de dos factores principales: el volumen de muestras de aprendizaje y la dimensionalidad del espacio de representación. Los costes relativos al volumen de muestras pueden reducirse si podemos aplicar técnicas de condensación de conjunto de datos. Para reducir el coste de la dimensionalidad del espacio de representación se ha propuesto el uso de procedimientos de selección de atributos que permiten reducir esta dimensionalidad de forma que produzcan los menores efectos perturbadores en el proceso de aprendizaje y en los posteriores de clasificación.

Esta cuestión conecta con el problema de la valoración de la representatividad de cada atributo, es decir, hasta qué punto es importante o accesorio, significativo o redundante. Para formalizar el problema de la Reducción de la Dimensionalidad, intentaremos expresarlo en la siguiente forma, para cada muestra i determinar una selección o transformación de atributos:

$$x_{ij} \rightarrow y_{ik} \quad j = 1, \dots, n; k = 1, \dots, l \quad l < n \quad (23.101)$$

Tal que transforma los datos originales $\mathbf{x} \in \mathbf{R}^n$ en los de salida $\mathbf{y} \in \mathbf{R}^l$ de menor dimensionalidad, $l < n$. Existen dos aproximaciones generalmente utilizadas para afrontar este problema:

- **Selección de Atributos.** En este caso los datos resultantes son extraídos desde el conjunto original formando un subconjunto del original. El gran problema de este enfoque es que es necesario determinar cuáles son los más significativos respecto a las clases del conjunto de datos [Theodoridis y Koutroumbas, 2003, cap. 5].
- **Transformación de Atributos.** En este caso se genera un transformación de los datos de forma que los resultantes pueden ser ordenados sistemáticamente en cuanto a su relevancia. Existen técnicas que simplifican este problema al generar conjuntos significativos en el número deseado, l , o bien conjuntos sistemáticamente ordenados en cuanto a relevancia.

Para resolver el primer enfoque es necesario disponer de la función de distribución probabilística de los datos, lo cual no siempre es posible. Analizaremos la segunda opción referente a generar transformaciones de los datos que posibilitan una selección sistemática. Consideraremos varias técnicas frecuentemente utilizadas en la comunidad de Reconocimiento de Formas, como son el Análisis de Componentes Principales(PCA), el de Componentes Independientes(ICA) y el de Escalado Multidimensional (MDS). Otras técnicas como las de mapas autoorganizables también resuelven este problema, pero su metodología es más apropiada para contenidos de Redes Neuronales.

23.6.1 Análisis de componentes principales

El procedimiento de análisis de componentes principales (Principal Component Analysis, PCA) es uno de los más utilizados en tareas de análisis de datos por su simplicidad computacional y robustez matemática. Consiste en generar una transformación lineal en los atributos, de forma que se dé lugar a un conjunto final que sea escalable en cuanto a representar la variabilidad inicial de los datos. El objetivo del análisis de componentes principales es determinar un nuevo sistema de coordenadas en las que expresar un conjunto de datos tal que exista independencia en cuanto a sus estadísticos de segundo orden. La media o estadístico de primer orden se obtiene como:

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_{ij} \quad (23.102)$$

La matriz de covarianza, o estadístico de segundo orden, se obtiene como:

$$\Sigma_{ab} = \frac{1}{m} \sum_{i=1}^m (x_{ia} - \mu_a)(x_{ib} - \mu_b) \quad (23.103)$$

La técnica PCA se basa en determinar las componentes independientes de la matriz de covarianza mediante la obtención de sus autovalores en la forma siguiente:

$$(\Sigma - \lambda \mathbf{I})\mathbf{U} = 0 \quad (23.104)$$

donde \mathbf{I} es la matriz identidad, λ es un autovalor de Σ y \mathbf{U} un autovector. Existen técnicas algebraicas, que no describiremos aquí, que permiten obtener eficientemente los autovectores y autovalores. Este conjunto de los autovalores forma una sucesión de números positivos decreciente: $\lambda_1 > \lambda_2 > \dots > \lambda_n$, que también son decrecientes en cuanto a la capacidad para representar la variabilidad de los datos originales. Básicamente la técnica PCA consiste en determinar las direcciones virtuales en las cuales la variabilidad de los datos es máxima y posteriormente realizar una rotación de los ejes del sistema de coordenadas hasta que los nuevos coincidan con estas direcciones virtuales. Para cada autovalor λ_a existe un autovector \mathbf{U}_a de coordenadas U_{ia} que verifica las siguientes propiedades:

$$\mathbf{U}_a^T \mathbf{U}_b = \delta_{ab} \quad (23.105)$$

Es decir, que el conjunto de autovectores que constituyen las bases del espacio transformado sea un sistema ortonormal de vectores. La transformación de los datos que se propone es de la forma lineal siguiente, también denominada **Transformada de Karhunen-Loéve** (KL):

$$y_k = \sum_{j=1}^n U_{jk} (x_j - \mu_j) \quad k = 1, \dots, n \quad (23.106)$$

Esta transformación incluye dos operaciones distintas. Por un lado, una traslación de los ejes hasta el centroide de los datos, lo que genera unos ejes centrados en el conjunto de datos. Por otro lado, una rotación de los ejes determinada por los autovectores. En la práctica no se eligen todos los autovectores, sino un subconjunto relevante de dimensionalidad $l < n$, de forma que:

$$\underbrace{\lambda_1 > \lambda_2 > \dots > \lambda_l}_{l} > \dots > \lambda_n \quad (23.107)$$

Ello determina la siguiente transformación de los datos con reducción de la dimensionalidad:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{pmatrix} = \begin{pmatrix} U_{11} & \dots & U_{1n} \\ U_{21} & \dots & U_{2n} \\ \vdots & \ddots & \vdots \\ U_{l1} & \dots & U_{ln} \end{pmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ \vdots \\ x_n - \mu_n \end{pmatrix} \quad (23.108)$$

Una implementación minimalista en Matlab de esta transformación es la siguiente que genera el conjunto de datos transformado de uno dado:

```
function Y = PCA(X,1)
    [V,D] = eig(cov(X));
    [lamb,index] = sort(diag(D), 'descend');
    Y = (X-repmat(mean(X,1),m,1))*V(:,index(1:l));
end
```

23.6.2 Análisis de componentes independientes

El objetivo del análisis de componentes independientes (Independent Component Analysis, ICA) es generar una transformación de los datos tal que el resultado esté formado por atributos que sean estadísticamente independientes. En PCA los atributos no son estadísticamente independientes, sino que sólo lo son respecto a los estadísticos de segundo orden. Por esta razón ICA es más general que PCA, pero evidentemente más complejo. La forma práctica de realizar la conversión es básicamente la de una máquina lineal:

$$y_k = f\left(\sum_{j=1}^n w_{kj}x_j + w_{k0}\right) \quad k = 1, \dots, n \quad (23.109)$$

donde $f()$ es una función de activación derivable. Las incógnitas del proceso son los pesos w que se elegirán, de tal forma que las variables de salida sean estadísticamente independientes. Esto impone que las variables de salida deban maximizar o minimizar una cierta función objetivo que podremos formular matemáticamente en función de los pesos incógnitas. La técnica es en este sentido muy similar a otros problemas ya estudiados de Reconocimiento de Formas. Es bien conocido por Teoría de la Información que cuando un conjunto de variables son independientes la entropía de las mismas es máxima. Recurrirremos pues a formular la cantidad de información de las variables de salida, para lo cual necesitamos su distribución probabilística. La entropía de una variable multidimensional \mathbf{x} en función de su distribución de probabilidad es:

$$H(\mathbf{x}) = E[-\ln p(\mathbf{x})] = -\int p(\mathbf{x}) \ln p(\mathbf{x}) d\mathbf{x} \quad (23.110)$$

En nuestro caso tenemos dos sistemas de variables las originales \mathbf{x} y las transformadas \mathbf{y} , de tal forma que la distribución de probabilidades en el segundo caso está determinada por:

$$p_{\mathbf{y}}(\mathbf{y}) = \frac{p_{\mathbf{x}}(\mathbf{x})}{|\mathbf{J}|} \quad (23.111)$$

donde \mathbf{J} es el Jacobiano de la transformación de coordenadas:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_n}{\partial x_n} \end{pmatrix} \quad (23.112)$$

que depende de los pesos de la transformación en la forma siguiente:

$$\frac{\partial y_i}{\partial x_j} = w_{ij} \frac{df(u)}{du} = \beta w_{ij} (1 - y_i^2) \quad (23.113)$$

Obsérvese que el último término $(1 - y_i^2)$ es constante en la línea del índice i y por tanto puede ser extraído como factor común en el cálculo del determinante del Jacobiano, que puede expresarse como:

$$|\mathbf{J}| = \beta^n |\mathbf{w}| \prod_{i=1}^n (1 - y_i^2) \quad (23.114)$$

La entropía de las variables de salida se puede expresar como:

$$H(\mathbf{y}) = -E[\ln p_{\mathbf{y}}(\mathbf{y})] = E[\ln |\mathbf{J}|] - E[\ln p_{\mathbf{x}}(\mathbf{x})] \quad (23.115)$$

donde el último término puede ignorarse, dado que es la entropía de los datos de entrada que es constante y no depende de la transformación. Pero, puesto que el Jacobiano (es decir, la transformación de las coordenadas) no depende de la distribución de probabilidad, su esperanza estadística es:

$$H(\mathbf{y}) = -E[\ln |\mathbf{J}|] = \ln |\mathbf{J}| = n \ln \beta + \ln |\mathbf{w}| + \sum_{i=1}^n \ln(1 - y_i^2) \quad (23.116)$$

El objetivo es pues maximizar respecto a los pesos \mathbf{w} la siguiente expresión:

$$\max_{\mathbf{w}} \ln |\mathbf{J}| \quad (23.117)$$

Lo que se puede hacer habitual, por efecto del método del gradiente, en este tipo de problemas. Obsérvese que ICA al contrario que PCA es una transformación no lineal de coordenadas.

23.6.3 Escalado multidimensional

La técnica de Escalado Multidimensional (Multidimensional Scaling, MDS) aborda el problema de reducción de la dimensionalidad mediante un enfoque diferente. En este caso no existe transformación de coordenadas, sino que el objetivo es que las muestras en el espacio transformado guarden entre sí unas distancias que sean lo más parecidas a las distancias existentes entre ellas en el espacio original. Obsérvese que no es un método orientado a las coordenadas de los datos, sino a las distancias de las muestras. Este enfoque permite que la técnica sea más abstracta que las anteriores y pueda ser aplicada a situaciones de análisis de datos más generales donde se parte de matrices de interdistancias entre muestras, aunque las mismas carezcan de base vectorial.

Sea δ_{ij} la distancia en el espacio original, de mayor dimensionalidad, entre las muestras i y j . Sea d_{ij} la distancia de las mismas muestras proyectadas en un espacio de menor dimensionalidad, tal y como muestra la Figura 23.11, que trata de transformar o proyectar un conjunto de muestras desde un espacio de dimensión 3 hacia uno de dimensión 2. El objetivo que se pretende alcanzar es la determinación de las coordenadas de las muestras en el espacio final, tal que el error de distancias de las mismas entre ambos espacios sea lo menor posible. Obsérvese que el espacio original no necesita estar definido vectorialmente, tan sólo se necesita la matriz de distancias.

Las distancias en el espacio transformado expresadas en un sistema de coordenadas $\mathbf{y} \in \mathbf{R}^l$ que son virtuales y carecen de significado o semántica serán las siguientes:

$$d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\| \quad (23.118)$$

El procedimiento más utilizado de MDS es la técnica de Samon, que consiste en minimizar la siguiente función objetivo de error relativo entre las distancias:

$$\min_{\mathbf{y}} J(\mathbf{y}) = \frac{1}{\sum_j \sum_{i < j} \delta_{ij}} \sum_j \sum_{i < j} \frac{(d_{ij}(\mathbf{y}) - \delta_{ij})^2}{\delta_{ij}} \quad (23.119)$$

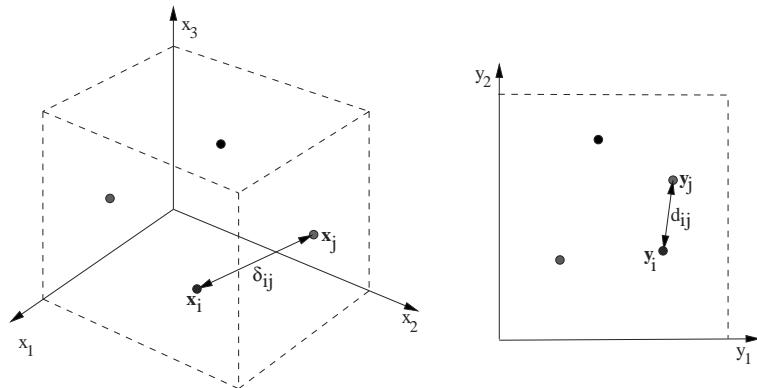


Figura 23.11: Escalado Multidimensional (MDS). Tipos de distancias en el espacio original \mathbf{R}^n y transformado \mathbf{R}^l , con $l < n$.

Obsérvese que la matriz δ_{ij} son datos y que la función objetivo depende de las coordenadas objetivo. Si existen m muestras, el número de incógnitas es $m \times l$. Para alcanzar la solución es necesario aplicar el método del gradiente a la función objetivo. Para ello se tiene que:

$$\nabla_{\mathbf{y}_k} J = \frac{2}{\sum_j \sum_{i < j} \delta_{ij}} \sum_{j \neq k} \frac{(d_{kj} - \delta_{kj})}{\delta_{kj}} \frac{\mathbf{y}_k - \mathbf{y}_j}{d_{kj}} \quad (23.120)$$

El sistema de coordenadas resultante carece de significado semántico, dado que es un espacio virtual. La solución obtenida no es única, sino que puede ser transformada por diversas operaciones geométricas que no alteren las distancias, tales como rotaciones y traslaciones arbitrarias.

23.7 Validación y comparación de clasificadores

En las secciones precedentes hemos presentado diferentes tipos de clasificadores y sus técnicas de entrenamiento. Una vez efectuado el aprendizaje del clasificador, es necesario realizar un conjunto de pruebas o test para poder conocer sus prestaciones. Estas pueden hacer referencia a los dos procesos de Reconocimiento de Formas que comentamos en la introducción de este capítulo. Por un lado tenemos la eficiencia del clasificador en línea, y por otro lado, la del proceso de entrenamiento. Ambos son importantes, pero la primera puede tener mayor relevancia, dado que el proceso de entrenamiento suele ser off-line. Podemos dividir la eficiencia en dos tipos. El primero hace referencia a la eficiencia computacional, es decir al consumo de recursos en términos de tiempo y memoria que requiere. El segundo tipo, el que nos interesa en esta sección, es el relativo a la tasa de error que comete el clasificador. Algunas de las cuestiones que deben ser resueltas a posteriori del entrenamiento de un clasificador son las siguientes:

- **Test** y evaluación de las prestaciones de un clasificador. Para ello se suele utilizar un conjunto de datos, que mediante diversas técnicas permita entrenar y evaluar sus prestaciones.
- **Comparación** de las prestaciones de diferentes clasificadores para determinar el más adecuado o para validar comparativamente las prestaciones de un nuevo clasificador respecto a otros existentes.
- **Combinación** de los resultados de diferentes clasificadores para obtener una clasificación más precisa. También suele utilizarse para obtener clasificaciones de mayor calidad a partir de varios clasificadores de baja calidad pero bajo coste.

El resultado de un clasificador es un diagnóstico y como mostramos en la introducción de este capítulo, el conjunto de diagnósticos es un conjunto cerrado y excluyente. En este sentido, la combinación de los resultados de diferentes clasificadores puede abordarse con la misma metodología que se utiliza en IA para tratar con la combinación de evidencias. Además, se pueden utilizar mecanismos simplistas como tomar el valor de clasificación mayoritario. También se pueden utilizar técnicas más avanzadas como las de **Fusión de Datos** [Web, 2002, sec. 8.4.] y **Mezcla de Expertos** [Duda y otros, 2001, sec. 9.7]

23.7.1 Test de clasificadores

Como colofón de todo proceso de entrenamiento de un clasificador se debe proceder a evaluar sus prestaciones, por ejemplo, en términos del error esperado. Partiendo de un conjunto de datos supervisado se trata de obtener una medida de ese error. Al exponer el clasificador bayesiano construimos una matriz de confusión $E_{i \rightarrow j}$ que proporcionaba información del grado de error del clasificador, relacionado con el entrelazado de las funciones de distribución. Para un caso discreto de N muestras, podemos construir una magnitud similar como: $N_{i \rightarrow j}$ que es el número de muestras de la clase Ω_i que son clasificadas como pertenecientes a la clase Ω_j . Para un sistema de c clases, el error $p(\epsilon)$ está relacionado con los elementos no diagonales:

$$p(\epsilon) = \frac{\sum_{i=1}^c \sum_{j \neq i} N_{i \rightarrow j}}{N} \quad N = \sum_{i=1}^c \sum_{j=1}^c N_{i \rightarrow j} \quad (23.121)$$

Para realizar estas evaluaciones es necesario distinguir entre el **conjunto de datos de entrenamiento** con el que se realiza el proceso de aprendizaje y el **conjunto de datos de prueba**, cuyas muestras son sometidas a clasificación y a partir del cual se construye la matriz de confusión.

El procedimiento de test más simple es el método de **Resustitución**, donde ambos conjuntos de entrenamiento y test son el mismo. En muchos sistemas de clasificación no es buena idea utilizar las muestras de entrenamiento como muestras de test, por cuanto el error estará falseado. Un ejemplo de este tipo son los de redes neuronales que se entrena para producir la respuesta correcta para un conjunto de muestras de entrenamiento. La posterior utilización de estas mismas muestras para verificar la

calidad del clasificador es contraproducente, producirá error nulo sistemáticamente. Igual fenómeno existe con el clasificador NN. No es el caso del clasificador k NN, donde el resultado de clasificar una muestra de entrenamiento puede no coincidir con el valor correcto de su clase.

Un procedimiento alternativo es el método **Holdout**, en cuyo caso el conjunto de datos original se divide en dos subconjuntos mutuamente excluyentes. Uno de ellos es utilizado como conjunto de entrenamiento y el otro como test. En este procedimiento queda por definir los tamaños y composición de ambos conjuntos. El resultado de la evaluación puede ser muy dependiente de estas elecciones. Aunque la selección aleatoria es la que menos desviación produce por la intervención humana, pueden existir grandes desviaciones de la estima del error.

Para resolver ese problema se utiliza una variante sistemática que no implica la toma de decisiones previas. Una variante de este método es la denominada **Validación Cruzada** (cross-validation, leave-one-out). En este caso con un conjunto de datos de m muestras procedemos a entrenar el clasificador con $m - 1$ muestras, dejando una fuera, de ahí su otra denominación; posteriormente, evaluamos la respuesta del clasificador para la muestra que quedó fuera. Este proceso se debe repetir m veces dejando fuera una muestra diferente en cada caso. Esto permite realizar una evaluación sistemática, pero enormemente costosa si el conjunto de datos es grande.

Una técnica que puede aplicarse previamente al de partición del conjunto de datos original es la conocida como **Bootstrap** que genera conjunto de datos aleatorios a partir del original. Un conjunto de datos bootstrap se crea mediante la selección aleatoria (con reemplazamiento) de m muestras desde un conjunto de datos original de justamente m muestras, lo que implica que existirán muestras repetidas en el conjunto de datos bootstrap. Este proceso se repite b veces y se evalúa las prestaciones de un clasificador sobre los b conjunto de datos generados, obteniéndose la media y varianza. Dada la naturaleza aleatoria de la selección, en cada una se excluirán algunas muestras generando una variedad de resultados. Una crítica que se puede hacer a este procedimiento es que al introducir muestras repetidas no existente en el conjunto de datos original, podrá provocar errores sistemáticos en el procedimiento posterior de evaluación.

La técnica **Bagging** (véase el capítulo 17), aunque también se presenta como una de combinación de clasificadores, es una técnica de estabilización de clasificadores poco robustos. Para ello, se toma un conjunto de datos original y se generan b datasets por la técnica de bootstrap. Se entrena un clasificador por cada uno de estos conjunto de datos y cuando se aplica en tareas on-line se toma como resultado del clasificador el mayoritario de los b resultados a la forma incógnita.

La técnica de **Boosting** (véase el capítulo 17) es similar en el sentido de ser aplicable a clasificadores débiles. Se basa en proporcionar un sistema de pesos a cada muestra que denota la importancia o relevancia de la misma. Los clasificadores deben ser tales que admitan el pesado de las muestras en los algoritmos de entrenamientos. **AdaBoost** (Adaptative Boosting) es la variante básica de este procedimiento. La técnica modifica iterativamente los pesos de las muestras, partiendo de una situación de igualdad, aumentando la importancia de las mal clasificadas, lo que tiende a reducir la tasa de error si el clasificador es sensible a estos pesos.

23.7.2 Comparación de prestaciones

Como resultado de la evaluación de clasificadores podríamos inferir resultados acerca de la calidad relativa entre ellos. Desgraciadamente, la existencia de ciertos teoremas limitativos impiden realizar una valoración de mejor calidad relativa con carácter general. El más conocido de estos es el teorema **No-Free-Lunch** (NFLT). Originalmente, NFLT es un teorema en el campo de optimización combinatoria desarrollado por Wolpert y Macready que establece que: **todos los algoritmos que buscan un extremo de una función objetivo, son igualmente buenos cuando son promediados sobre todas las posibles funciones objetivos**. De cuyo contenido podemos deducir varias lecturas posibles [Ho y Pepyne, 2002]:

1. Una estrategia de optimización de carácter general, es decir sin restricción de los problemas, no puede ser globalmente óptima o mejor que todas las demás, pero podrá ser especialmente óptima sólo si está especializada a un tipo concreto de problema.
2. En ausencia de cualquier conocimiento previo acerca de la función de optimización, todos los algoritmos tienen la mismas expectativas de prestaciones.

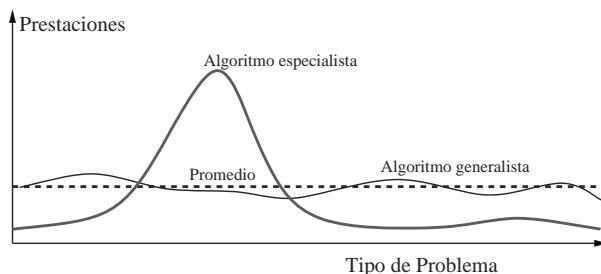


Figura 23.12: Teorema No-Free-Lunch sobre la igualdad promediada de todos los procedimientos clasificadores sobre la totalidad de dominios. La superioridad en un dominio específico se paga con su inferioridad en otros y, por tanto, con la falta de generalidad.

La aplicación de este principio a las técnicas de aprendizaje es inmediata. Si consideramos un algoritmo de aprendizaje como uno que busca una solución para una función objetivo, deberíamos concluir que no existe ningún procedimiento de aprendizaje que sea globalmente bueno para todas las funciones objetivos. Sin embargo, aquellas funciones objetivos que tienen interés en aprendizaje son de un conjunto restringido, por ejemplo las de mínimo error, por ello podemos dudar si podemos aplicar el sentido amplio del teorema a tareas de aprendizaje.

El resultado de este teorema también ha sido aplicado por Wolpert a problemas de comparación de algoritmos de aprendizaje supervisado. Las conclusiones obtenidas son que [Duda y otros, 2001, sec. 9.2]: **promediando sobre todos los conjuntos de datos posibles, la esperanza de error promedio de cualquier clasificador**

es la misma. Es decir que con carácter general, en ausencia de conocimiento o situación específica, todos los clasificadores tienen la misma calidad. Como conclusión, el diseño de clasificadores de superiores prestaciones en error de clasificación sólo puede establecerse en un dominio o aplicaciones específicas.

23.8 Ejercicios propuestos

Dado que la mayor parte de los procedimientos utilizados en Reconocimiento de Formas son de naturaleza matricial, Matlab es una herramienta idónea para las actividades formativas en esta disciplina. Se proponen algunas tareas sencillas para completar la formación. Para actividades de formación en posgrado e investigación en Reconocimiento de Formas, puede ser muy interesante utilizar librerías o toolboxes ampliamente utilizados, algunos ejemplo destacados son por un lado **Spider**¹ o también las **PRTTools**². Igualmente existe un texto con la implementación Matlab [Stork y Yom-Tov, 2004] de los algoritmos del texto fundamental en Reconocimiento de Formas de Duda y Hart en su nueva edición [Duda y otros, 2001].

Para fines de formación en la docencia de grado, o bien para estudio personal a nivel básico, es más adecuado la síntesis de las funciones más significativas. El resultado puede no ser tan eficiente computacionalmente, pero el valor formativo es considerablemente superior. Dada la cantidad de funciones que Matlab incorpora en sus toolboxes (los más útiles para Reconocimiento de Formas son los de Estadística, Redes Neuronales y Optimización) la tarea se puede simplificar y las prestaciones no desmerecen las versiones más profesionales.

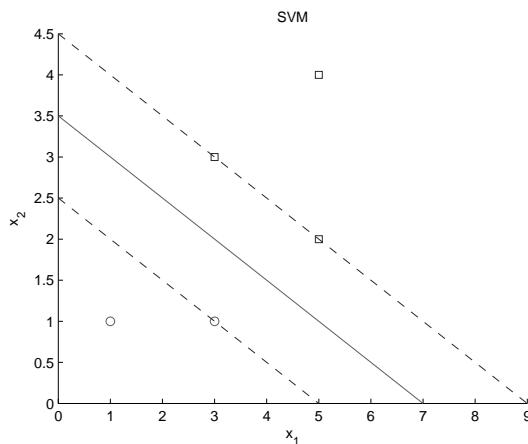


Figura 23.13: Ejemplo de conjunto de datos sencillo para implementar clasificadores lineales como el perceptón simple o la SVM.

¹<http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html>

²<http://www.prtools.org/>

Para ilustrar algunos de los problemas de Reconocimiento de Formas, se propone acceder al Repositorio de Aprendizaje **UCI Machine Learning Repository**³ [Newman y otros, 1998]. Por ejemplo, descargar los ficheros del conjunto de datos *Iris* de clasificación de flores que consta de cuatro atributos numéricos y tres clases de flores. Para facilitar su uso se propone editar localmente el fichero de muestras, por ejemplo reemplazando en la última columna de datos, los nombres de las clases de cada muestra por un número natural de código de la clase.

Como ejemplo simplificado de construcción de clasificadores consideremos el siguiente problema de dos clases. El conjunto de muestras está representado en la Figura 23.13. Las matrices de coordenadas de los puntos del conjunto de datos y de clases asociadas son:

$$\mathbf{X} = \begin{pmatrix} 1 & 1 \\ 3 & 1 \\ 5 & 2 \\ 3 & 3 \\ 5 & 4 \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \quad (23.122)$$

La matrix Ψ es de la forma siguiente:

$$\Psi = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ -1 & -5 & -2 \\ -1 & -3 & -3 \\ -1 & -5 & -4 \end{pmatrix} \quad (23.123)$$

Para obtener un clasificador lineal mediante un perceptrón simple, podríamos aplicar un procedimiento tan simple como:

```
function [a,test] = perceptronSimple(Psi)
    [m,n] = size(Psi);
    [a,fval,test]=fminunc(@(w)objetivo(w,Psi),rand(n,1));
end

function f = objetivo(a,Psi)
    f = norm(abs(psi*a)-psi*a)/2;
end
```

El número de soluciones es infinita, pero la SVM proporciona una solución más robusta. La aplicación del procedimiento visto en la página 1003 da como resultado:

$$\mathbf{a} = \begin{pmatrix} 3,5 \\ -0,5 \\ -1,0 \end{pmatrix} \quad \boldsymbol{\lambda} = \begin{pmatrix} 0,00 \\ 0,65 \\ 0,25 \\ 0,37 \\ 0,00 \end{pmatrix} \quad (23.124)$$

³<http://www.ics.uci.edu/~mlearn/MLRepository.html>

Teniendo en cuenta que $w_0 = a(1)$ y que $w_i = a(1 + i)$, este clasificador está representado por la frontera definida por la expresión:

$$3,5 - 0,5x_1 - 1,0x_2 = 0 \quad (23.125)$$

Si trasladamos la recta del clasificador hacia un lado y otro hasta que toquen con alguna muestra, podemos representar gráficamente la banda de seguridad. Las muestras que tocan estas bandas son la segunda, tercera y cuarta. Estas muestras son justamente aquellas que tienen los multiplicadores de Lagrange no nulos. Algunas tareas adicionales que puede realizar el lector son:

23.1. Codificar en Matlab una función para la lectura de conjunto de datos que sea invocada en la forma siguiente: $[X, C] = \text{leedataset}(\text{fichero})$, que proporciona dos matrices, X de dimensión $m \times n$ donde m es el número de muestras y n el de atributos y C de dimensión $m \times 1$ que contiene un número natural asociado a la clase de cada muestra. El argumento es el nombre del fichero que contienen los datos de un conjunto de datos.

23.2. La siguiente función Matlab realiza una operación de evaluación sobre un conjunto de datos determinado por sus matrices X y C . Identificar esa operación en términos de los contenidos teóricos expuestos en este capítulo.

```
function E = incognita(X,C)
    m = size(X,1);
    Y = X*X';
    dg = diag(Y);
    sq = repmat(dg,1,m)+repmat(dg',m,1)- 2*Y;
    for i=1:m, sq(i,i) = inf; end
    [d,nn] = min(sq,[],1);
    Ct = C(nn);
    E = length(find(Ct ~= C))/m;
end
```

23.3. Tratar de explicar el siguiente código Matlab de la versión dual de SVM, que hace uso de la función $\Psi = \text{ensambla}(X, C, k)$ que construye la matriz Ψ para un conjunto de datos dado y para una clase determinada por el número natural k y que se indica a continuación. Consultar los manuales del toolbox de Optimización.

```
function Psi = ensambla(X,C,k)
    [m,n] = size(X);
    I = ones(m,1);
    Y = [I, X];
    index = find(C ~= k),
    I(index) = - 1;
    Psi = repmat(I,[1,n+1]).*Y;
end
```

```
function [a,lambda,exitflag] = svm_dual(Psi)
y = Psi(:,1); [m,n] = size(Psi);
Z = Psi(:,2:end); H = Z*Z';
[lambda,fval,exitflag] = ...
    quadprog(H,-ones(m,1),[],[],y',0,zeros(m,1),[],rand(m,1));
w = lambda'*Z;
index = find(lambda > 0); k = index(1);
a = [(1-w'*Z(k,:))/y(k);w];
end
```

Referencias

- Optimization Toolbox User's Guide.* The MatWorks Inc., 2006.
- BISHOP, CHRISTOPHER M.: *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.
- CRISTIANI, N. y TAYLOR, J. S.: *An Introduction to Support Vector Machines.* Cambridge Univ. Press, 2000.
- DEVROYE, L.; GYORFI, L. y LUGOSI, G.: *A Probabilistic Theory of Pattern Recognition.* Springer, 1996.
- DUDA, R. O.; HART, P. E. y STORK, D. G.: *Pattern Classification.* John Wiley & Sons, 2001.
- FUKUNAGA, K.: *Statistical Pattern Recognition.* Morgan Kaufmann, 1990.
- HAYKIN, SIMON: *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1998.
- HO, Y.C. y PEPYNE, D.L.: «Simple Explanation of the No-Free-Lunch Theorem and Its Applications». *Journal of Optimization Theory and Applications*, 2002, **115(3)**, pp. 549–570.
- NEWMAN, D.J.; HETTICH, S.; BLAKE, C.L. y MERZ, C.J.: «UCI Repository of machine learning databases», 1998.
<http://www.ics.uci.edu/~mlearn/MLRepository.html>
- STORK, D. G. y YOM-TOV, E.: *Computer Manual in MATLAB to Accompany Pattern Classification.* John Wiley & Sons, 2004.
- THEODORIDIS, S. y KOUTROUMBAS, K.: *Pattern Recognition.* Elsevier, 2003.
- VAPNIK, V.N.: *The Nature of Statistical Learning Theory.* Springer, 2000.
- WEB, A.: *Statistical Pattern Recognition.* Jhon Wiley & Sons, 2002.
- WINSTON, W. L.: *Investigación de Operaciones. Aplicaciones y Algoritmos.* Thomson, 2005.

Recientemente se han cumplido 50 años del nacimiento de la Inteligencia Artificial (IA), período en el que se ha convertido en una de las áreas más prolíficas en cuanto a resultados de investigación. No podemos obviar que muchos de los desarrollos que surgieron de grupos de investigación en IA están presentes en muchas de las actividades que comúnmente desarrollamos, desde los sistemas de navegación de los automóviles, hasta los pequeños controladores inteligentes de algunos electrodomésticos. Tampoco podemos obviar los avances que se han producido en campos como la robótica que han supuesto una revolución en los métodos de fabricación. Toda esta labor investigadora ha producido gran cantidad de técnicas y metodologías que han dado lugar a numerosas disciplinas que dibujan un panorama que resulta casi imposible de abordar en un solo libro.

En Inteligencia Artificial: técnicas, métodos y aplicaciones, se han intentado conjugar los tópicos clásicos de la IA, que se vienen cubriendo en la docencia de grado, con otros tópicos avanzados que pueden ser ubicados tanto en la docencia de grado como de posgrado. De esta forma, el libro se ha dividido en las siguientes partes: Introducción, donde se analizan los diferentes paradigmas en los que se basa la IA en la actualidad; Representación de conocimiento e inferencia, donde se abordan desde los aspectos clásicos de representación de conocimiento y técnicas clásicas de razonamiento, hasta aspectos más avanzados relacionados con la gestión de la incertidumbre; Técnicas, entre las que se incluyen las clásicas de búsqueda hasta aspectos más avanzados como la computación evolutiva; Tareas, donde se analiza procesos complejos como pueden ser la planificación y el diagnóstico; Aprendizaje y minería de datos, donde se hace una introducción a algunas técnicas de aprendizaje supervisado y no supervisado, y finalmente, Aspectos metodológicos y aplicaciones, en la que se ha intentado abordar temas como la Ingeniería del conocimiento, Sistemas multiagentes y Razonamiento basado en casos.

Cada uno de los capítulos ha sido desarrollado por expertos de contrastada reputación en su campo, lo que ha permitido, no sólo exponer los aspectos básicos relacionados con el capítulo de una forma clara, sino que se ha podido dar una visión más global de cada tema indicando referencias bibliográficas con las que profundizar en cada uno de los tópicos.



Este libro dispone de OLC, Online Learning Center, página web asociada, lista para su uso inmediato y creada expresamente para facilitar la labor docente del profesor y el aprendizaje de los alumnos.
Se incluyen contenidos adicionales al libro y recursos para la docencia.

www.mhe.es/marin1