

Intelligenza artificiale

• **Un approccio moderno**
Volume 1

Quarta edizione

Stuart Russell, Peter Norvig
a cura di Francesco Amigoni

MyLab Codice per accedere
alla piattaforma



Pearson

Intelligenza artificiale

Un approccio moderno

Volume 1

Intelligenza artificiale

Un approccio moderno

Volume 1

Quarta edizione

Stuart Russell, Peter Norvig

a cura di Francesco Amigoni



© 2021 Pearson Italia, Milano - Torino

Authorized translation from the English language edition, entitled ARTIFICIAL INTELLIGENCE: A MODERN APPROACH, 4th Edition by STUART RUSSELL; PETER NORVIG, published by Pearson Education, Inc, publishing as Pearson, Copyright © 2020.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Italian language edition published by Pearson Italia S.p.A., Copyright © 2021.

Per i passi antologici, per le citazioni, per le riproduzioni grafiche, cartografiche e fotografiche appartenenti alla proprietà di terzi, inseriti in quest'opera, l'editore è a disposizione degli aventi diritto non potuti reperire nonché per eventuali non volute omissioni e/o errori di attribuzione nei riferimenti.

È vietata la riproduzione, anche parziale o ad uso interno didattico, con qualsiasi mezzo, non autorizzata.

Le fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941, n. 633.

Le riproduzioni effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da CLEAREDì, Corso di Porta Romana 108, 20122 Milano, e-mail autorizzazioni@clearedi.org e sito web www.clearedi.org.

I nostri libri sono ecosostenibili: la carta è prodotta sostenendo il ciclo naturale e per ogni albero tagliato ne viene piantato un altro; il cellofan è realizzato con plastiche da recupero ambientale o riciclate; gli inchiostri sono naturali e atossici; i libri sono prodotti in Italia e l'impatto del trasporto è ridotto al minimo.

Curatore per l'edizione italiana: Francesco Amigoni

Traduzione e redazione: Infostudio

Impaginazione: TOTEM di Andrea Astolfi

Grafica di copertina: Simone Tartaglia

Immagine di copertina: sebastien decoret/123RF

Stampa: Arti Grafiche Battaia – Zibido San Giacomo (MI)

Tutti i marchi citati nel testo sono di proprietà dei loro detentori.

9788891904454

Printed in Italy

4^a edizione: settembre 2021

Ristampa

00 01 02 03 04

Anno

21 22 23 24 25

LIBRI DI TESTO E SUPPORTI DIDATTICI

Il sistema di gestione per la qualità della Casa Editrice è certificato in conformità alla norma UNI EN ISO 9001:2015 per l'attività di progettazione, realizzazione e commercializzazione di: • prodotti editoriali scolastici, dizionari lessicografici, prodotti per l'editoria di varia ed universitaria • materiali didattici multimediali off-line • corsi di formazione e specializzazione in aula, a distanza, e-learning.

Member of CISQ Federation



La struttura dell'opera

Volume				
		1	2	Cap
Parte I Intelligenza artificiale	✓ MyLab	1	Introduzione	
	✓ MyLab	2	Agenti intelligenti	
Parte II Risoluzione di problemi	✓	3	Risolvere i problemi con la ricerca	
	✓	4	Ricerca in ambienti complessi	
Parte III Conoscenza, ragionamento e pianificazione	✓	5	Ricerca con avversari e giochi	
	✓	6	Problemi di soddisfacimento di vincoli	
Parte IV Conoscenza incerta e ragionamento in condizioni di incertezza	✓	7	Agenti logici	
	✓	8	Logica del primo ordine	
Parte V Apprendimento automatico	✓	9	Inferenza nella logica del primo ordine	
	✓	10	Rappresentazione della conoscenza	
Parte VI Comunicazione, percezione e azione	✓	11	Pianificazione automatica	
	✓	12	Quantificare l'incertezza	
Parte VII Conclusioni	✓	13	Ragionamento probabilistico	
	✓	14	Ragionamento probabilistico nel tempo	
Appendici	✓	15	Programmazione probabilistica	
	✓	16	Decisioni semplici	
Bibliografia	✓	17	Decisioni complesse	
	✓	18	Decisioni multiagente	
Indice analitico	✓	19	Apprendimento da esempi	
	✓	20	Apprendimento di modelli probabilistici	
Parte VII Conclusioni	✓	21	Deep learning	
	✓	22	Apprendimento per rinforzo	
Parte VI Comunicazione, percezione e azione	✓	23	Elaborazione del linguaggio naturale	
	✓	24	Deep learning per elaborazione del linguaggio naturale	
Appendici	✓	25	Visione artificiale	
	✓	26	Robotica	
Bibliografia	MyLab	✓	27	Filosofia, etica e sicurezza dell'intelligenza artificiale
	MyLab	✓	28	Futuro dell'intelligenza artificiale
Indice analitico	✓	✓	A	Basi matematiche
	✓	✓	B	Cenni sui linguaggi e sugli algoritmi

Nota dell'Editore

L'edizione italiana presenta, rispetto a quella inglese, alcune importanti modifiche quali la suddivisione dell'opera originale – veramente encyclopedica – in due volumi, e la parziale riorganizzazione strutturale degli argomenti presentati. Lo schema qui sopra riportato illustra sinteticamente le caratteristiche della nostra pubblicazione: i segni di spunta indicano in quale dei due volumi è presente ciascun capitolo; come è possibile vedere, i Capitoli 1-2 sono presenti nel Volume 1 e nella piattaforma MyLab, i Capitoli 27-28 sono presenti nel Volume 2 e nella piattaforma MyLab, mentre le Appendici sono presenti in entrambi i volumi e la Bibliografia è presente solo nella piattaforma MyLab. Sono nati così due testi, autonomi e autoconsistenti, che rendono non solo più agevole la consultazione ma consentono anche una migliore fruibilità dei contenuti sia da parte degli studenti (che trovano gli argomenti strutturati secondo l'organizzazione di molti corsi di laurea di primo e secondo livello) sia da parte dei professionisti che vogliono estendere le conoscenze al di fuori dal proprio campo specialistico.

Crediamo, in questo modo, di fornire un prezioso contributo per promuovere la conoscenza, la ricerca e la passione nei confronti di una disciplina così vasta e affascinante come l'intelligenza artificiale.

Sommario

Prefazione all’edizione italiana	XXI
Prefazione	XXIII
Gli autori	XXVI
Pearson MyLab	XXVII

PARTE 1 INTELLIGENZA ARTIFICIALE

1

Capitolo 1 Introduzione	3
1.1 Cos’è l’intelligenza artificiale?	4
1.1.1 Agire umanamente: l’approccio del test di Turing	4
1.1.2 Pensare umanamente: l’approccio della modellazione cognitiva	5
1.1.3 Pensare razionalmente: l’approccio delle “leggi del pensiero”	5
1.1.4 Agire razionalmente: l’approccio degli agenti razionali	6
1.1.5 Macchine che portano benefici	7
1.2 I fondamenti dell’intelligenza artificiale	8
1.2.1 Filosofia	8
1.2.2 Matematica	10
1.2.3 Economia	12
1.2.4 Neuroscienze	13
1.2.5 Psicologia	15
1.2.6 Ingegneria informatica	17
1.2.7 Teoria del controllo e cibernetica	18
1.2.8 Linguistica	19
1.3 La storia dell’intelligenza artificiale	19
1.3.1 La gestazione dell’intelligenza artificiale (1943–1956)	20
1.3.2 Primi entusiasmi, grandi aspettative (1952–1969)	21
1.3.3 Una dose di realtà (1966–1973)	23
1.3.4 Sistemi esperti (1969–1986)	24

1.3.5 Il ritorno delle reti neurali (1986–presente)	26
1.3.6 Ragionamento probabilistico e apprendimento automatico (1987–presente)	27
1.3.7 Big data (2001–presente)	28
1.3.8 Deep learning (2011–presente)	29
1.4 Lo stato dell’arte	30
1.5 Rischi e opportunità dell’intelligenza artificiale	33
1.6 Riepilogo	37
Note storiche e bibliografiche	38

Capitolo 2 Agenti intelligenti	39
---------------------------------------	-----------

2.1 Agenti e ambienti	39
2.2 Comportarsi correttamente: il concetto di razionalità	42
2.2.1 Misure di prestazione	42
2.2.2 Razionalità	43
2.2.3 Onniscienza, apprendimento e autonomia	43
2.3 La natura degli ambienti	45
2.3.1 Specificare un ambiente	45
2.3.2 Proprietà degli ambienti operativi	47
2.4 La struttura degli agenti	50
2.4.1 Programmi agente	50
2.4.2 Agenti reattivi semplici	52
2.4.3 Agenti reattivi basati su modello	54
2.4.4 Agenti basati su obiettivi	56
2.4.5 Agenti basati sull’utilità	57
2.4.6 Agenti capaci di apprendere	58
2.4.7 Funzionamento dei componenti dei programmi agente	60
2.5 Riepilogo	62
Note storiche e bibliografiche	63

PARTE 2 RISOLUZIONE DI PROBLEMI

65

Capitolo 3 Risolvere i problemi con la ricerca	67
3.1 Agenti risolutori di problemi	68
3.1.1 Problemi di ricerca e soluzioni	69
3.1.2 La formulazione dei problemi	70
3.2 Problemi esemplificativi	71
3.2.1 Problemi standardizzati	71
3.2.2 Problemi reali	73
3.3 Algoritmi di ricerca	75
3.3.1 Ricerca best-first	77
3.3.2 Strutture dati per la ricerca	77
3.3.3 Cammini ridondanti	79
3.3.4 Misurare le prestazioni nella risoluzione di problemi	80
3.4 Strategie di ricerca non informata	81
3.4.1 Ricerca in ampiezza	81
3.4.2 Algoritmo di Dijkstra o ricerca a costo uniforme	82
3.4.3 Ricerca in profondità e problema della memoria	83
3.4.4 Ricerca a profondità limitata e ad approfondimento iterativo	85
3.4.5 Ricerca bidirezionale	87
3.4.6 Confronto tra le strategie di ricerca non informata	88
3.5 Strategie di ricerca informata o euristica	88
3.5.1 Ricerca best-first greedy o “golosa”	88
3.5.2 Ricerca A*	90
3.5.3 Confini di ricerca	94
3.5.4 Ricerca soddisfacente: euristiche inammissibili e ricerca A* pesata	95
3.5.5 Ricerca con memoria limitata	97
3.5.6 Ricerca euristica bidirezionale	101
3.6 Funzioni euristiche	102
3.6.1 Effetto dell’accuratezza dell’euristica sulle prestazioni	103
3.6.2 Generare euristiche da problemi rilassati	104
3.6.3 Generare euristiche da sottoproblemi: database di pattern	106
3.6.4 Generare euristiche con punti di riferimento (<i>landmark</i>)	107
3.6.5 Imparare a cercare meglio	108
3.6.6 Apprendere euristiche dall’esperienza	109
3.7 Riepilogo	109
Note storiche e bibliografiche	111

Capitolo 4 Ricerca in ambienti complessi	115
4.1 Ricerca locale e problemi di ottimizzazione	116
4.1.1 Ricerca hill climbing	116
4.1.2 Simulated annealing	119
4.1.3 Ricerca local beam	120
4.1.4 Algoritmi evolutivi	121
4.2 Ricerca locale in spazi continui	124
4.3 Ricerca con azioni non deterministiche	127
4.3.1 Il mondo dell'aspirapolvere erratico	127
4.3.2 Alberi di ricerca AND-OR	128
4.3.3 Prova, prova ancora	130
4.4 Ricerca con osservazioni parziali	131
4.4.1 Ricerca in assenza di osservazioni	131
4.4.2 Ricerca in ambienti parzialmente osservabili	135
4.4.3 Risoluzione di problemi parzialmente osservabili	137
4.4.4 Un agente per ambienti parzialmente osservabili	137
4.5 Agenti per ricerca online e ambienti sconosciuti	140
4.5.1 Problemi di ricerca online	140
4.5.2 Agenti per ricerca online	142
4.5.3 Ricerca locale online	143
4.5.4 Apprendimento nella ricerca online	146
4.6 Riepilogo	146
Note storiche e bibliografiche	147
Capitolo 5 Ricerca con avversari e giochi	151
5.1 La teoria dei giochi	152
5.1.1 Giochi a somma zero a due giocatori	152
5.2 Decisioni ottime nei giochi	154
5.2.1 L'algoritmo di ricerca minimax	155
5.2.2 Decisioni ottime nei giochi multiplayer	156
5.2.3 Potatura alfa-beta	157
5.2.4 Ordinamento delle mosse	159
5.3 Ricerca alfa-beta euristica	161
5.3.1 Funzioni di valutazione	162
5.3.2 Tagliare la ricerca	163
5.3.3 Potatura in avanti	165
5.3.4 Ricerca su albero e ricerca in tabelle	166

5.4	Ricerca ad albero Monte Carlo	167
5.5	Giochi stocastici	170
5.5.1	Funzioni di valutazione per giochi con elementi casuali	172
5.6	Giochi parzialmente osservabili	173
5.6.1	Kriegspiel: scacchi parzialmente osservabili	174
5.6.2	Giochi di carte	176
5.7	Limitazioni degli algoritmi di ricerca per i giochi	178
5.8	Riepilogo	180
	Note storiche e bibliografiche	180

Capitolo 6 Problemi di soddisfacimento di vincoli **185**

6.1	Definizione dei problemi di soddisfacimento di vincoli	186
6.1.1	Un problema di esempio: colorazione di una mappa	186
6.1.2	Un problema di esempio: programmazione di lavori	187
6.1.3	Varianti del formalismo CSP	188
6.2	Propagazione di vincoli: inferenza nei CSP	191
6.2.1	Consistenza di nodo	191
6.2.2	Consistenza d'arco	191
6.2.3	Consistenza di cammino	193
6.2.4	K -consistenza	193
6.2.5	Vincoli globali	194
6.2.6	Sudoku	195
6.3	Ricerca con backtracking per CSP	196
6.3.1	Ordinamento di variabili e valori	198
6.3.2	Alternanza di ricerca e inferenza	199
6.3.3	Backtracking intelligente: guardarsi indietro	200
6.3.4	Apprendimento dei vincoli	202
6.4	Ricerca locale per CSP	202
6.5	La struttura dei problemi	204
6.5.1	Condizionamento con insieme di taglio	205
6.5.2	Scomposizione ad albero	207
6.5.3	Simmetria di valore	208
6.6	Riepilogo	209
	Note storiche e bibliografiche	209

PARTE 3 CONOSCENZA, RAGIONAMENTO E PIANIFICAZIONE

213

Capitolo 7 Agenti logici

215

7.1	Agenti basati sulla conoscenza	216
7.2	Il mondo del wumpus	218
7.3	Logica	221
7.4	Logica proposizionale: una logica molto semplice	224
7.4.1	Sintassi	224
7.4.2	Semantica	225
7.4.3	Una semplice base di conoscenza	227
7.4.4	Una semplice procedura di inferenza	227
7.5	Dimostrazione di teoremi nella logica proposizionale	228
7.5.1	Inferenza e dimostrazioni	230
7.5.2	Dimostrazione per risoluzione	232
7.5.3	Clausole di Horn e clausole definite	236
7.5.4	Concatenazione in avanti e concatenazione all'indietro	237
7.6	Model checking proposizionale efficiente	239
7.6.1	Un algoritmo di backtracking completo	240
7.6.2	Algoritmi di ricerca locale	242
7.6.3	Il panorama dei problemi SAT casuali	243
7.7	Agenti basati sulla logica proposizionale	244
7.7.1	Lo stato corrente del mondo	245
7.7.2	Un agente ibrido	248
7.7.3	Stima dello stato con la logica	248
7.7.4	Costruzione di piani mediante inferenza proposizionale	250
7.8	Riepilogo	253
	Note storiche e bibliografiche	254

Capitolo 8 Logica del primo ordine

257

8.1	Ancora sulla rappresentazione	257
8.1.1	Il linguaggio del pensiero	258
8.1.2	Mettere insieme il meglio dei linguaggi formali e naturali	260
8.2	Sintassi e semantica della logica del primo ordine	262
8.2.1	Modelli per la logica del primo ordine	262

8.2.2 Simboli e interpretazioni	264
8.2.3 Termini	266
8.2.4 Formule atomiche	266
8.2.5 Formule complesse	267
8.2.6 Quantificatori	267
8.2.7 Uguaglianza	270
8.2.8 Semantica dei database	271
8.3 Usare la logica del primo ordine	272
8.3.1 Asserzioni e query nella logica del primo ordine	272
8.3.2 Il dominio della parentela	273
8.3.3 Numeri, insiemi e liste	274
8.3.4 Il mondo del wumpus	276
8.4 Ingegneria della conoscenza nella logica del primo ordine	278
8.4.1 Il processo di ingegneria della conoscenza	278
8.4.2 Il dominio dei circuiti elettronici	280
8.5 Riepilogo	283
Note storiche e bibliografiche	284

Capitolo 9 Inferenza nella logica del primo ordine **287**

9.1 Inferenza proposizionale e inferenza del primo ordine	287
9.1.1 Riduzione all'inferenza proposizionale	288
9.2 Unificazione e inferenza del primo ordine	289
9.2.1 Unificazione	291
9.2.2 Memorizzazione e recupero di informazioni	292
9.3 Concatenazione in avanti	294
9.3.1 Clausole definite del primo ordine	294
9.3.2 Un semplice algoritmo di concatenazione in avanti	295
9.3.3 Concatenazione in avanti efficiente	297
9.4 Concatenazione all'indietro	301
9.4.1 Un algoritmo di concatenazione all'indietro	301
9.4.2 Programmazione logica	302
9.4.3 Inferenza ridondante e cicli infiniti	303
9.4.4 Semantica dei database di Prolog	305
9.4.5 Programmazione logica a vincoli	306
9.5 Risoluzione	306
9.5.1 Forma normale congiuntiva per la logica del primo ordine	307
9.5.2 La risoluzione come regola di inferenza	308

9.5.3 Alcuni esempi di dimostrazione	309
9.5.4 Completezza della risoluzione	311
9.5.5 L'uguaglianza	314
9.5.6 Strategie di risoluzione	315
9.6 Riepilogo	318
Note storiche e bibliografiche	318

Capitolo 10 Rappresentazione della conoscenza **323**

10.1 Ingegneria ontologica	324
10.2 Categorie e oggetti	326
10.2.1 Composizione fisica	328
10.2.2 Misure	329
10.2.3 Oggetti: cose e roba	331
10.3 Eventi	332
10.3.1 Tempo	333
10.3.2 Fluenti e oggetti	334
10.4 Oggetti mentali e logica modale	335
10.4.1 Altre logiche modali	337
10.5 Sistemi di ragionamento per categorie	338
10.5.1 Reti semantiche	338
10.5.2 Logiche descrittive	340
10.6 Ragionare con informazione di default	342
10.6.1 Circoscrizione e logica di default	342
10.6.2 Sistemi di mantenimento della verità	344
10.7 Riepilogo	346
Note storiche e bibliografiche	347

Capitolo 11 Pianificazione automatica **353**

11.1 Definizione di pianificazione classica	354
11.1.1 Esempio: trasporto aereo di merci	355
11.1.2 Esempio: il problema della ruota di scorta	356
11.1.3 Esempio: il mondo dei blocchi	356
11.2 Algoritmi per la pianificazione classica	358
11.2.1 Ricerca in avanti nello spazio degli stati per problemi di pianificazione	358
11.2.2 Ricerca all'indietro per pianificazione	359

11.2.3 Pianificazione come soddisfacibilità booleana	360
11.2.4 Altri approcci di pianificazione classica	361
11.3 Euristiche per la pianificazione	362
11.3.1 Potatura indipendente dal dominio	363
11.3.2 Astrazione di stato nella pianificazione	365
11.4 Pianificazione gerarchica	366
11.4.1 Azioni di alto livello	366
11.4.2 Ricerca di soluzioni primitive	368
11.4.3 Ricerca di soluzioni astratte	369
11.5 Pianificazione e azione in ambienti non deterministici	374
11.5.1 Pianificazione senza sensori	376
11.5.2 Pianificazione condizionale	380
11.5.3 Pianificazione online	381
11.6 Tempo, scheduling e risorse	384
11.6.1 Rappresentazione di vincoli temporali e di risorse	384
11.6.2 Risoluzione di problemi di scheduling	385
11.7 Analisi degli approcci alla pianificazione	388
11.8 Riepilogo	388
Note storiche e bibliografiche	389
PARTE 4 CONOSCENZA INCERTA E RAGIONAMENTO IN CONDIZIONI DI INCERTEZZA	395
Capitolo 12 Quantificare l'incertezza	397
12.1 Agire in condizioni di incertezza	397
12.1.1 Riassumere l'incertezza	398
12.1.2 Incertezza e decisioni razionali	399
12.2 Notazione base della teoria della probabilità	401
12.2.1 A che cosa fanno riferimento le probabilità	401
12.2.2 Il linguaggio delle proposizioni nelle asserzioni probabilistiche	403
12.2.3 Gli assiomi della teoria della probabilità e la loro ragionevolezza	405
12.3 Inferenza basata su distribuzioni congiunte complete	407
12.4 Indipendenza	410

12.5	La regola di Bayes e il suo utilizzo	411
12.5.1	Applicare la regola di Bayes: il caso semplice	412
12.5.2	Usare la regola di Bayes: combinazione di evidenze	413
12.6	Modelli di Bayes ingenui	414
12.6.1	Classificazione di testi con un modello Bayes ingenuo	415
12.7	Il mondo del wumpus rivisitato	416
12.8	Riepilogo	419
	Note storiche e bibliografiche	420

Capitolo 13 Ragionamento probabilistico **423**

13.1	Rappresentazione della conoscenza in un dominio incerto	423
13.2	La semantica delle reti bayesiane	426
	Un metodo per costruire reti bayesiane	427
	Compattezza e ordinamento dei nodi	428
13.2.1	Relazioni di indipendenza condizionale nelle reti bayesiane	430
13.2.2	Rappresentazione efficiente delle distribuzioni condizionate	431
13.2.3	Reti bayesiane con variabili continue	433
13.2.4	Caso di studio: assicurazione auto	436
13.3	Inferenza esatta nelle reti bayesiane	438
13.3.1	Inferenza per enumerazione	439
13.3.2	L'algoritmo di eliminazione delle variabili	441
13.3.3	La complessità dell'inferenza esatta	444
13.3.4	Algoritmi di clustering	446
13.4	Inferenza approssimata nelle reti bayesiane	447
13.4.1	Metodi di campionamento diretto	447
13.4.2	Inferenza mediante simulazione con catene di Markov	453
13.4.3	Compilare inferenza approssimata	460
13.5	Reti causalì	461
13.5.1	Rappresentare le azioni: l'operatore <i>do</i>	462
13.5.2	Il criterio back-door	464
13.6	Riepilogo	465
	Note storiche e bibliografiche	465

Capitolo 14 Ragionamento probabilistico nel tempo	471
14.1 Tempo e incertezza	472
14.1.1 Stati e osservazioni	472
14.1.2 Modelli di transizione e modelli sensoriali	473
14.2 Inferenza nei modelli temporali	475
14.2.1 Filtraggio e predizione	476
14.2.2 Smoothing	479
14.2.3 Trovare la sequenza più probabile	481
14.3 Modelli di Markov nascosti	483
14.3.1 Algoritmi semplificati basati su matrici	484
14.3.2 Un esempio di modello di Markov nascosto: localizzazione	486
14.4 Filtri di Kalman	489
14.4.1 Aggiornare le distribuzioni gaussiane	490
14.4.2 Un semplice esempio monodimensionale	491
14.4.3 Il caso generale	493
14.4.4 Applicabilità del filtraggio di Kalman	494
14.5 Reti bayesiane dinamiche	496
14.5.1 Costruire le DBN	496
14.5.2 Inferenza esatta nelle DBN	500
14.5.3 Inferenza approssimata nelle DBN	501
14.6 Riepilogo	507
Note storiche e bibliografiche	508
Capitolo 15 Programmazione probabilistica	511
15.1 Modelli relazionali di probabilità	512
15.1.1 Sintassi e semantica	513
15.1.2 Esempio: valutare il livello di abilità dei giocatori	516
15.1.3 Inferenza nei modelli relazionali di probabilità	517
15.2 Modelli probabilistici a universo aperto	518
15.2.1 Sintassi e semantica	519
15.2.2 Inferenza nei modelli probabilistici a universo aperto	521
15.2.3 Esempi	522
15.3 Tenere traccia di un mondo complesso	526
15.3.1 Esempio: tracciamento multitarget	526
15.3.2 Esempio: monitoraggio del traffico	529

15.4	Programmi come modelli probabilistici	530
15.4.1	Esempio: lettura di testo	530
15.4.2	Sintassi e semantica	530
15.4.3	Risultati delle inferenze	532
15.4.4	Migliorare il programma generativo per incorporare un modello di Markov	533
15.4.5	Inferenza nei programmi generativi	533
15.5	Riepilogo	534
	Note storiche e bibliografiche	535

Capitolo 16 Decisioni semplici **539**

16.1	Combinare credenze e desideri in condizioni di incertezza	540
16.2	Le basi della teoria dell'utilità	541
16.2.1	Vincoli su preferenze razionali	541
16.2.2	Le preferenze razionali conducono all'utilità	543
16.3	Funzioni di utilità	544
16.3.1	Valutazione e scale di utilità	544
16.3.2	L'utilità del denaro	545
16.3.3	Utilità attesa e delusione dopo la decisione	547
16.3.4	Giudizio umano e irrazionalità	549
16.4	Funzioni di utilità multiattributo	551
16.4.1	Dominanza	552
16.4.2	Struttura delle preferenze e utilità multiattributo	554
16.5	Reti di decisione	556
16.5.1	Rappresentare un problema con una rete di decisione	556
16.5.2	Valutazione delle reti di decisione	558
16.6	Il valore dell'informazione	558
16.6.1	Un semplice esempio	558
16.6.2	Una formula generale per l'informazione perfetta	559
16.6.3	Proprietà del valore dell'informazione	561
16.6.4	Implementazione di un agente che raccoglie informazioni	561
16.6.5	Raccolta di informazioni non miope	562
16.6.6	Analisi di sensibilità e decisioni robuste	563
16.7	Preferenze ignote	564
16.7.1	Incertezza sulle proprie preferenze	564
16.7.2	Delega agli esseri umani	565
16.8	Riepilogo	568
	Note storiche e bibliografiche	568

Capitolo 17 Decisioni complesse	573
17.1 Problemi di decisione sequenziali	573
17.1.1 Utilità nel tempo	576
17.1.2 Politiche ottime e utilità degli stati	578
17.1.3 Scale di ricompensa	580
17.1.4 Rappresentazione di MDP	581
17.2 Algoritmi per MDP	583
17.2.1 Iterazione dei valori	583
17.2.2 Iterazione delle politiche	587
17.2.3 Programmazione lineare	589
17.2.4 Algoritmi online per MDP	589
17.3 Problemi dei banditi	591
17.3.1 Calcolo dell'indice di Gittins	594
17.3.2 Il problema dei banditi di Bernoulli	595
17.3.3 Politiche quasi ottime per problemi dei banditi	595
17.3.4 Varianti non indicizzabili	596
17.4 MDP parzialmente osservabili	598
17.4.1 Definizione di POMDP	599
17.5 Algoritmi per risolvere POMDP	601
17.5.1 Iterazione dei valori per POMDP	601
17.5.2 Algoritmi online per POMDP	604
17.6 Riepilogo	606
Note storiche e bibliografiche	606
Capitolo 18 Decisioni multiagente	609
18.1 Proprietà degli ambienti multiagente	609
18.1.1 Un solo decisore	609
18.1.2 Decisori multipli	610
18.1.3 Pianificazione multiagente	611
18.1.4 Pianificazione con più agenti: cooperazione e coordinamento	614
18.2 Teoria dei giochi non cooperativi	615
18.2.1 Giochi con una sola mossa: giochi in forma normale	615
18.2.2 Benessere sociale	619
18.2.3 Giochi ripetuti	623
18.2.4 Giochi sequenziali: la forma estesa	627
18.2.5 Payoff incerti e giochi di assistenza	633

18.3	Teoria dei giochi cooperativi	635
18.3.1	Strutture di coalizioni e risultati	636
18.3.2	Strategia nei giochi cooperativi	637
18.3.3	Computazioni nei giochi cooperativi	639
18.4	Decisioni collettive	642
18.4.1	Assegnare compiti con il contract net protocol	642
18.4.2	Allocare risorse scarse con le aste	643
18.4.3	Votazione	648
18.4.4	Contrattazione	650
18.5	Riepilogo	654
	Note storiche e bibliografiche	655

Capitolo 27 Filosofia, etica e sicurezza dell'intelligenza artificiale Online

Capitolo 28 Futuro dell'intelligenza artificiale Online

Appendice A Basi matematiche **659**

A.1	Analisi di complessità e notazione O()	659
A.1.1	Analisi asintotica	659
A.1.2	NP e problemi intrinsecamente difficili	660
A.2	Vettori, matrici e algebra lineare	661
A.3	Distribuzioni di probabilità	663
	Note storiche e bibliografiche	665

Appendice B Cenni sui linguaggi e sugli algoritmi **667**

B.1	Definire i linguaggi con la forma di Backus-Naur (BNF)	667
B.2	Descrivere gli algoritmi con lo pseudocodice	668

Bibliografia Online

Indice analitico 671

Prefazione all'edizione italiana

L'intelligenza artificiale (IA) sta vivendo una stagione di grande crescita. Nell'industria, il ruolo dell'IA è determinante: secondo l'Artificial Intelligence Index Report 2021 della Stanford University, più della metà delle aziende operanti nel mondo impiega qualche sistema di IA, con percentuali che si alzano per le aziende che operano nei settori tecnologici, automobilistici e della finanza. In campo accademico, il numero di articoli scientifici sui temi dell'IA è in costante aumento e, sempre secondo lo stesso rapporto, nel 2020 ha raggiunto quasi il 4% di tutte le pubblicazioni scientifiche. Crescono anche i corsi universitari sull'IA, sia a livello di laurea triennale sia a livello di laurea magistrale. Inoltre, l'IA è diventata centrale nel dibattito pubblico, per le sue enormi potenzialità e per i rischi connessi allo sviluppo e all'impiego di sistemi che possono avere effetti rilevanti sulle vite di miliardi di persone. Possiamo dire che, a oltre 60 anni dalla sua nascita, l'IA suscita attenzione e curiosità come non mai e si pone come una disciplina moderna, effervescente e interessante da approfondire e da studiare.

Il nuovo aggiornamento della versione italiana, che si allinea alla recente quarta edizione, del libro di Russell e Norvig si inserisce in questo quadro e rappresenta una opportunità per i docenti e gli studenti universitari e per i lettori non specialisti, che più difficilmente hanno accesso alla letteratura in lingua inglese. La dimensione enciclopedica permette al libro di soddisfare sia le esigenze di una divulgazione approfondita sia quelle didattiche, fornendo, da un lato, un utilissimo riferimento per il professionista o il semplice curioso che vuole approfondire, all'interno di un quadro concettuale coerente, le basi teoriche dell'IA o gli sviluppi più recenti di una particolare tematica e, dall'altro lato, un testo strutturato per il docente che deve organizzare un corso universitario (e per gli studenti che lo seguono).

Dal punto di vista didattico, è apprezzabile lo sforzo degli autori di fornire una visione unificata dell'IA, che ha vissuto una crescita spesso tumultuosa, a volte guidata più dall'eccezionalizzazione della scoperta che da un rigido programma di ricerca. Tale visione unificata si concretizza nell'idea centrale di agente, intorno a cui ruotano le presentazioni dei diversi argomenti, da quelli classici a quelli più innovativi, e nel tentativo di fare emergere concetti trasversali alle varie sotto-discipline che spesso "parlano" linguaggi diversi. Inoltre, la fitta ragnatela di collegamenti e rimandi fra le diverse parti del testo rinforza l'idea di una disciplina che sta consolidando una sua piena identità teorica.

Dal punto di vista della divulgazione approfondita, gli autori sono molto attenti nel presentare non solo le tecniche più aggiornate per affrontare i vari problemi, ma anche gli impegni più rilevanti di queste tecniche nelle applicazioni reali, da cui emerge un quadro in cui i metodi dell'IA sono ormai parte di innumerevoli sistemi di uso industriale e quotidiano.

Un aspetto che è importante sottolineare è la scelta di non trascurare, accanto alla discussione tecnica, gli aspetti filosofici ed etici che sono legati alla natura stessa della disciplina e che la caratterizzano, in modo solo apparentemente incongruente, da molto prima della sua nascita. Pur con le ovvie semplificazioni, la loro trattazione arricchisce il contesto della presentazione fornendo al lettore gli strumenti per ragionare criticamente sul contenuto tecnico-scientifico dell'IA.

A mio avviso la conoscenza e la consapevolezza, anche da parte dei non specialisti, dei principi di funzionamento dei sistemi di IA costituisce il primo passo per governare l'impatto che tali sistemi stanno avendo sulle nostre attività e per individuare e mitigare i rischi deri-

vanti. Da questo punto di vista, il testo di Russell e Norvig offre un ottimo strumento per favorire l'impostazione condivisa e consapevole di un percorso di regolamentazione, che ormai appare ineludibile.

All'inizio degli anni 1970, Bertram Raphael espresse l'idea che l'IA accomuna i problemi che non sappiamo ancora come risolvere con un computer. Marco Somalvico, uno dei primi a contribuire alla diffusione dell'IA in Italia, ripeteva spesso la paradossale conseguenza che, una volta che un problema di IA è stato risolto, non fa più parte dell'IA. La situazione è decisamente cambiata rispetto a questi inizi pionieristici: attualmente la disciplina ha solide basi scientifiche e molti ricadute applicative. La nuova edizione del libro di Russell e Norvig fornisce una panoramica moderna sui fondamenti teorici e sui metodi dell'IA e suggerisce i confini e le possibilità di una disciplina che, per fortuna, non ha perso del tutto il carattere rivoluzionario delle origini.

Francesco Amigoni
Politecnico di Milano

Prefazione¹

L'intelligenza artificiale (IA) è un campo molto vasto, e questo è un libro ponderoso. Abbiamo cercato di presentare l'intero panorama della disciplina, che racchiude la logica, la probabilità e la matematica del continuo; la percezione, il ragionamento, l'apprendimento e l'azione; l'equità, la fiducia, il bene sociale e la sicurezza, oltre ad applicazioni che spaziano dai dispositivi microelettronici ai robot per l'esplorazione planetaria, fino ai servizi online con miliardi di utenti.

Il sottotitolo di questo volume è “Un approccio moderno”. Il senso è che abbiamo scelto di presentare gli argomenti da un punto di vista attuale. Abbiamo sintetizzato tutti i temi in un contesto comune, rivedendo i primi lavori che risalgono alle origini della disciplina attraverso le idee e la terminologia che sono prevalenti oggi. Chiediamo scusa agli specialisti dei singoli campi dell'IA se, in questo modo, le loro specifiche aree di ricerca dovessero risultare meno riconoscibili.

Novità di questa edizione

Questa edizione riflette i cambiamenti avvenuti nel campo dell'IA dopo la pubblicazione dell'edizione precedente.

- Ci concentriamo maggiormente sull'apprendimento automatico (*machine learning*) anziché sull'ingegneria della conoscenza, perché oggi vi è una maggiore disponibilità di dati, risorse di calcolo e nuovi algoritmi.
- Deep learning, programmazione probabilistica e sistemi multiagente sono argomenti trattati più in dettaglio; a ognuno di essi è dedicato un intero capitolo.
- La trattazione di sistemi per la comprensione del linguaggio naturale, la robotica e la visione artificiale è stata rivista per riflettere l'impatto del deep learning.
- Il capitolo dedicato alla robotica ora tratta anche i robot che interagiscono con gli esseri umani e l'applicazione alla robotica dell'apprendimento per rinforzo.
- Nell'edizione precedente avevamo definito come obiettivo dell'IA la creazione di sistemi che puntano a massimizzare l'utilità attesa, in cui la specifica informazione sull'utilità – l'obiettivo – è fornita dai progettisti umani. Ora non ipotizziamo più che l'obiettivo sia fissato e noto al sistema di IA, che invece può essere incerto sui veri obiettivi degli esseri umani per conto dei quali opera. Il sistema deve apprendere che cosa massimizzare e deve funzionare in modo appropriato anche quando è incerto sull'obiettivo.
- Abbiamo approfondito la trattazione dell'impatto dell'IA sulla società, considerando anche i temi fondamentali dell'etica, dell'equità, della fiducia e della sicurezza.
- Gli esercizi non sono più riportati al termine di ogni capitolo, ma sono presenti in lingua originale sul sito <http://aima.cs.berkeley.edu/>. In questo modo potranno essere continuamente aggiornati, espansi e migliorati, per soddisfare le esigenze dei docenti e per riflettere i progressi più recenti compiuti nel campo dell'IA e nei software correlati. Per quanto

¹ Abbiamo scelto di riportare integralmente in queste pagine il testo della Prefazione dell'edizione originale – che fornisce una panoramica sulla struttura dell'opera e sul contenuto di tutti i capitoli – per offrire al lettore una visione complessiva del testo di Stuart Russell e Peter Norvig. L'edizione italiana è stata suddivisa in due volumi, come dettagliato a pagina V (N.d.E.).

riguarda l'edizione italiana si è deciso di tradurre i testi degli esercizi che sono reperibili nella piattaforma MyLab a corredo del testo.

- Nel complesso, circa il 25% dei contenuti è interamente nuovo, e il rimanente 75% è stato ampiamente rivisto e riscritto per presentare un quadro più uniforme della disciplina. Il 22% delle citazioni in questa edizione fa riferimento a opere pubblicate dopo il 2010.

Una visione d'insieme

Il principale tema unificante è l'idea di **agente intelligente**. Nella nostra definizione, l'IA è lo studio degli agenti che ricevono percezioni dall'ambiente ed eseguono azioni. Ogni agente implementa una funzione che mette in corrispondenza sequenze percettive e azioni, e il nostro scopo è presentare diverse tecniche per rappresentare tali funzioni: per esempio gli agenti reattivi, i pianificatori in tempo reale, i sistemi basati sulla teoria delle decisioni e i sistemi di deep learning. Focalizziamo l'attenzione sull'apprendimento sia come metodo di costruzione per sistemi competenti, sia come modo per estendere il campo d'azione del progettista in territori sconosciuti. La robotica e la visione non sono trattati come problemi indipendenti, ma nella loro funzione al servizio del raggiungimento degli obiettivi. Viene inoltre posto l'accento sull'importanza dell'ambiente operativo nel determinare l'architettura di agente più appropriata.

Il nostro scopo principale è trasmettere le *idee* emerse negli ultimi settant'anni di ricerca nel campo dell'IA e nei due precedenti millenni di pensiero. Abbiamo cercato di evitare eccessivi formalismi nella presentazione dei concetti, mantenendo tuttavia la precisione. Per dare maggiore concretezza alle idee esposte, abbiamo incluso formule matematiche e algoritmi in pseudocodice; i concetti e le notazioni matematiche sono descritti nell'Appendice A, mentre lo pseudocodice è descritto nell'Appendice B.

Il libro è principalmente rivolto a un corso o a una serie di corsi universitari; ha 28 capitoli, ognuno dei quali richiede circa una settimana di lezioni, perciò in tutto richiede due semestri. Un corso di un solo semestre può utilizzare capitoli selezionati secondo gli interessi del docente e degli studenti. Il libro può anche essere adottato in un corso di dottorato (eventualmente integrato con alcune delle fonti principali suggerite nelle note bibliografiche), o per studiare in modo autonomo, o come riferimento.

In tutto il libro, i *punti importanti* sono evidenziati da una lente d'ingrandimento al margine. Ogni volta che un **termine** nuovo è definito per la prima volta, è riportato anche a margine, in modo da facilitarne il ritrovamento. Se il **termine** è utilizzato in modo significativo nel seguito, viene ancora riportato in grassetto, ma non a margine.

L'unico prerequisito è la familiarità con i concetti di base dell'informatica (algoritmi, strutture dati, complessità) a livello dei primi anni di studi universitari. Conoscenze di analisi matematica e algebra lineare (a livello del primo anno) sono utili per alcuni degli argomenti.

Risorse online

Esercizi, progetti di programmazione e progetti di ricerca non sono più riportati al termine di ciascun capitolo, ma sono soltanto online.

- Nella piattaforma **MyLab** abbinata all'edizione italiana del volume si trovano i testi in italiano degli esercizi relativi a ciascun capitolo, oltre alla bibliografia estesa.

Nel sito web dell'edizione originale, aima.cs.berkeley.edu, sono disponibili le seguenti risorse.

- Esercizi, progetti di programmazione e progetti di ricerca. Sono fornite istruzioni per trovare gli esercizi utilizzando il nome o l'argomento.
- Implementazioni degli algoritmi descritti nel libro in Python, Java e altri linguaggi di programmazione (al momento si trovano presso github.com/aimacode).

- Materiali supplementari e collegamenti per studenti e docenti.
- Istruzioni su come segnalare eventuali errori nel libro, nella probabile eventualità che ve ne siano.

Ringraziamenti

Per creare un libro serve un villaggio globale. Oltre 600 persone hanno letto parti del libro e hanno fornito suggerimenti per migliorarlo; siamo grati a tutte loro. L'elenco completo è disponibile online presso aima.cs.berkeley.edu/ack.html. Per motivi di spazio, ci limitiamo a citare qui soltanto le persone che hanno fornito contributi particolarmente importanti. Innanzitutto coloro che hanno scritto alcune parti:

- Judea Pearl (Paragrafo 13.5, Reti causali);
- Vikash Mansinghka (Paragrafo 15.4, Programmi come modelli probabilistici);
- Michael Wooldridge (Capitolo 18, Decisioni multiagente);
- Ian Goodfellow (Capitolo 21, Deep learning);
- Jacob Devlin e Mei-Wing Chang (Capitolo 24, Deep learning per elaborazione del linguaggio naturale);
- Jitendra Malik e David Forsyth (Capitolo 25, Visione artificiale);
- Anca Dragan (Capitolo 26, Robotica).

E poi alcune persone che hanno svolto un ruolo fondamentale:

- Cynthia Yeung e Malika Cantor (project management);
- Julie Sussman e Tom Galloway (copyediting e suggerimenti per la scrittura);
- Omari Stephens (illustrazioni);
- Tracy Johnson (editor);
- Erin Ault e Rose Kernan (copertina e conversione colori);
- Nalin Chhibber, Sam Goto, Raymond de Lacaze, Ravi Mohan, Ciaran O'Reilly, Amit Patel, Dragomir Radiv e Samagra Sharma (sviluppo di codice online e mentoring);
- Studenti della Google Summer of Code (sviluppo di codice online).

Stuart desidera ringraziare sua moglie, Loy Sheflott, per l'infinita pazienza e la saggezza senza limiti. Si augura che Gordon, Lucy, George e Isaac leggeranno presto questo libro, dopo che lo avranno perdonato per aver lavorato così a lungo su di esso. Gli studenti del RUGS (Russell's Unusual Group of Students) sono stati insolitamente utili, come sempre.

Peter desidera ringraziare i suoi genitori (Torsten e Gerda) per averlo avviato agli studi e sua moglie (Kris), i figli (Bella e Juliet), i colleghi, il capo e gli amici per averlo incoraggiato e sopportato nelle lunghe ore passate a scrivere e a riscrivere.

Gli autori

Stuart Russell è nato nel 1962 a Portsmouth, in Inghilterra. Si è laureato in fisica *cum laude* alla Oxford University nel 1982 e ha ottenuto il dottorato in informatica a Stanford nel 1986. In seguito è passato alla University of California a Berkeley, dove è professore (e in precedenza direttore) di informatica, direttore del Center for Human-Compatible AI e titolare della cattedra Smith-Zadeh in ingegneria. Nel 1990 ha ricevuto il Presidential Young Investigator Award della National Science Foundation e nel 1995 ha conseguito il Computers and Thought Award. È Fellow dell'American Association for Artificial Intelligence, dell'Association for Computing Machinery e dell'American Association for the Advancement of Science, Honorary Fellow del Wadham College, a Oxford, e Andrew Carnegie Fellow. È stato titolare della cattedra Blaise Pascal a Parigi dal 2012 al 2014. Ha pubblicato oltre 300 articoli su una vasta gamma di argomenti di intelligenza artificiale. Tra gli altri suoi libri vi sono *The Use of Knowledge in Analogy and Induction*, *Do the Right Thing: Studies in Limited Rationality* (con Eric Wefald) e *Human Compatible: Artificial Intelligence and the Problem of Control*.

Peter Norvig è Director of Research presso Google ed è stato il direttore responsabile per gli algoritmi di ricerca web. È stato co-docente di un corso online sull'IA a cui si sono scritti 160.000 studenti e che ha dato un forte impulso al filone dei *mooc* (*massive open online classes*). È stato a capo della Computational Sciences Division presso l'Ames Research Center della NASA, dove era supervisore delle attività di ricerca e sviluppo in intelligenza artificiale e robotica. Ha conseguito la laurea in matematica applicata alla Brown University e un dottorato in informatica a Berkeley. È stato docente all'University of Southern California e professore a Berkeley e Stanford. È Fellow dell'American Association for Artificial Intelligence, dell'Association for Computing Machinery, dell'American Academy of Arts and Sciences e della California Academy of Science. Tra gli altri libri che ha scritto vi sono *Paradigms of AI Programming: Case Studies in Common Lisp* e *Verbmob: A Translation System for Face-to-Face Dialog* e *Intelligent Help Systems for UNIX*.

I due autori hanno condiviso il premio AAAI/EAAI Outstanding Educator nel 2016.

Pearson MyLab

UN AMBIENTE PER LO STUDIO

L'attività di apprendimento continua in **MyLab**, l'**ambiente digitale per lo studio** che completa il libro offrendo **risorse didattiche** fruibili sia in **modo autonomo** sia per **assegnazione del docente**. Il **codice sulla copertina** di questo libro consente l'**accesso per 18 mesi a MyLab**.

COME ACCEDERE

- 1.** **Registrati** come **studente universitario** all'indirizzo registrazione.pearson.it (se sei già registrato passa al punto successivo);
- 2.** effettua il **login** alla tua MyPearsonPlace all'indirizzo www.pearson.it/place e registra il prodotto digitale cliccando su **Attiva prodotto** ed inserendo il codice presente in copertina;
- 3.** entra nella sezione **Prodotti** e clicca sul tasto **AVVIA** presente di fianco all'immagine della copertina del testo;
- 4.** clicca su **classe MyLab studio autonomo** o, in alternativa, su **Iscriviti a una classe** ed inserisci il codice classe indicato dal tuo docente.



CHE COSA CONTIENE

MyLab offre la possibilità di accedere al Manuale online: l'**edizione digitale del testo** arricchita da funzionalità che permettono di personalizzarne la fruizione, attivare la sintesi vocale, inserire segnalibri.

Inoltre la **piattaforma digitale MyLab** integra e monitora il percorso individuale di studio con **attività formative e valutative specifiche**. La loro descrizione dettagliata è consultabile nella pagina di catalogo dedicata al libro, all'indirizzo link.pearson.it/2B45DA29 oppure tramite il presente QR code.



P A R T E

1

Intelligenza artificiale

Capitolo 1 Introduzione

Capitolo 2 Agenti intelligenti



Introduzione

- 1.1 Cos'è l'intelligenza artificiale?
- 1.2 I fondamenti dell'intelligenza artificiale
- 1.3 La storia dell'intelligenza artificiale
- 1.4 Lo stato dell'arte
- 1.5 Rischi e opportunità dell'intelligenza artificiale
- 1.6 Riepilogo
Note storiche e bibliografiche

Dove cerchiamo di spiegare perché consideriamo l'intelligenza artificiale un argomento degno di studio accurato, e cerchiamo anche di decidere che cos'è precisamente, essendo questa una buona cosa da fare prima di cominciare.

Gli esseri umani fanno riferimento a se stessi con il termine *homo sapiens* perché ritengono che la loro **intelligenza** sia molto importante. Per migliaia d'anni abbiamo cercato di comprendere come pensiamo e agiamo; ovvero, come il nostro cervello, un semplice mucchio di materia, può percepire, capire, predire e manipolare un mondo molto più grande e complicato. Il campo dell'**intelligenza artificiale**, o IA, va ancora più in là: il suo obiettivo non è solo comprendere, ma anche *costruire* entità intelligenti, cioè macchine in grado di calcolare come agire in modo efficace e sicuro in un'ampia varietà di situazioni nuove.

L'IA risulta in tutti i sondaggi uno dei campi di ricerca più interessanti e maggiormente in crescita, e ha già generato fatturati per oltre mille miliardi di dollari. L'esperto di IA Kai-Fu Lee prevede che il suo impatto sarà “maggiore di qualsiasi altra cosa nella storia dell'umanità”. Inoltre, i confini intellettuali dell'IA sono ancora molto aperti. Uno studente di una scienza tradizionale come la fisica potrebbe pensare che tutte le idee migliori siano già state formulate da Galileo, Newton, Curie, Einstein e gli altri, mentre l'IA ha ancora molti posti liberi nella lista delle sue migliori menti.

Al giorno d'oggi l'IA è suddivisa in un grande numero di sottodiscipline: alcune aree, come l'apprendimento e la percezione, sono generali e trasversali; altre invece si occupano di problemi specifici, come il gioco degli scacchi, la dimostrazione di teoremi matematici, la scrittura di poesie, la guida di automobili e la diagnosi di malattie. L'IA si può applicare a ogni sfera del pensiero umano; è un campo davvero universale.

1.1 Cos’è l’intelligenza artificiale?

razionalità

Abbiamo affermato che l’IA è una disciplina interessante, ma non abbiamo ancora detto *cos’è*. In passato gli studiosi hanno indagato diverse versioni di IA. Alcuni hanno definito l’intelligenza in termini di fedeltà alla prestazione *umana*, mentre altri preferiscono una definizione formale di intelligenza come **razionalità**, o per dirla in parole povere, “fare la cosa giusta”. D’altro canto, alcuni considerano l’intelligenza una proprietà dei *processi di pensiero* e del *ragionamento*, mentre altri si concentrano sul *comportamento* intelligente, con una caratterizzazione esterna.¹

Dalle due dimensioni di umano versus razionale² e pensiero versus comportamento si ottengono quattro possibili combinazioni, per ognuna delle quali sono stati sviluppati programmi di ricerca con il sostegno di diversi gruppi di studiosi. I metodi usati sono necessariamente diversi: l’approccio che persegue un’intelligenza simile a quella umana deve essere in parte una scienza empirica correlata alla psicologia, richiedendo osservazioni e ipotesi riguardo il comportamento umano e i processi di pensiero. Un approccio razionalista, invece, sfrutta una combinazione di matematica e ingegneria e si collega alla statistica, alla teoria del controllo e all’economia. I vari gruppi di studiosi si sono duramente contestati l’un l’altro, ma anche aiutati. Ora presenteremo i quattro approcci nei dettagli.

test di Turing

1.1.1 Agire umanamente: l’approccio del test di Turing

Il **test di Turing**, proposto da Alan Turing nel 1950, è stato concepito come un esperimento mentale in grado di evitare la vaghezza filosofica della domanda: “Una macchina è in grado di pensare?”. Un computer supererà il test se un esaminatore umano, dopo aver posto alcune domande in forma scritta, non sarà in grado di capire se le risposte provengono da una persona o no. Il Capitolo 27 discute i dettagli del test e valuta se un computer in grado di passarlo può essere davvero ritenuto intelligente. Per adesso ci limitiamo a notare che programmare una macchina in grado di superare il test applicato in modo rigoroso richiede un sacco di lavoro. Il computer avrebbe bisogno delle seguenti capacità:

interpretazione del linguaggio naturale

- **interpretazione del linguaggio naturale** per comunicare con successo nel linguaggio umano;
- **rappresentazione della conoscenza** per memorizzare quello che conosce o sente;
- **ragionamento automatico** per rispondere alle domande e trarre nuove conclusioni;
- **apprendimento automatico** (*machine learning*) per adattarsi a nuove circostanze, individuare ed estrapolare schemi.

rappresentazione della conoscenza

ragionamento automatico

apprendimento automatico

test di Turing totale

Turing riteneva che la simulazione *fisica* di una persona non fosse richiesta per dimostrare l’intelligenza. Tuttavia, esiste anche un cosiddetto **test di Turing totale** che richiede l’interazione con oggetti e persone nel mondo reale. Per superare il test di Turing totale, un robot necessiterà anche di:

visione artificiale
robotica

- **visione artificiale** e riconoscimento vocale per percepire il mondo;
- **robotica** per manipolare gli oggetti e spostarsi fisicamente.

¹ Agli occhi del pubblico generico si fa talvolta confusione tra i termini “intelligenza artificiale” (IA) e “apprendimento automatico” o “machine learning”. L’apprendimento automatico è un ramo dell’intelligenza artificiale che studia la capacità di migliorare le prestazioni basandosi sull’esperienza. Alcuni sistemi di IA usano metodi di apprendimento automatico per raggiungere livelli di competenza richiesti, altri no.

² Non stiamo suggerendo che gli esseri umani siano letteralmente “irrazionali” nel senso che siano “privi di lucidità mentale”, stiamo semplicemente osservando che non sempre le decisioni umane sono matematicamente perfette.

Le sei discipline elencate precedentemente abbracciano gran parte dell'IA. Tuttavia, i ricercatori non hanno dedicato molti sforzi al tentativo di costruire un sistema capace di superare il test di Turing, ritenendo più importante studiare i principî alla base dell'intelligenza: dopotutto, la ricerca del "volo artificiale" ha raggiunto il successo quando ingegneri e inventori hanno smesso di imitare gli uccelli e hanno iniziato a utilizzare le gallerie del vento e a studiare l'aerodinamica. I manuali di ingegneria aerospaziale non definiscono l'obiettivo della loro disciplina come la creazione di "macchine che volano esattamente come un piccione, in modo così perfetto da ingannare anche gli altri piccioni".

1.1.2 Pensare umanamente: l'approccio della modellazione cognitiva

Se vogliamo dire che un programma pensa come un essere umano, dobbiamo prima determinare come noi pensiamo. Ci sono tre modi per farlo:

- l'**introspezione**, ovvero il tentativo di catturare "al volo" i nostri pensieri mentre scorrono;
- la **sperimentazione psicologica**, ovvero l'osservazione di una persona in azione;
- l'**imaging cerebrale**, ovvero l'osservazione del cervello in azione.

introspezione
sperimentazione
psicologica
imaging cerebrale

Una volta che abbiamo formulato una teoria della mente sufficientemente precisa, diventa possibile esprimere la sotto forma di un programma per computer. Se il comportamento del software, per quanto riguarda il suo input/output, corrisponde a quello di una persona, potrebbe essere una prova che alcuni dei meccanismi del programma operano anche negli esseri umani. Per esempio, Allen Newell e Herbert Simon, che hanno sviluppato il GPS o General Problem Solver (Newell e Simon, 1961), non si accontentarono semplicemente di scrivere un programma che risolvesse correttamente i problemi: il loro vero interesse consisteva nel confronto della sequenza e della temporizzazione dei passi del suo ragionamento con quelli osservati in soggetti umani. Il campo interdisciplinare delle **scienze cognitive** unisce modelli computazionali sviluppati dall'IA e tecniche di sperimentazione psicologica nel tentativo di costruire teorie precise e verificabili sul funzionamento della mente umana.

scienze cognitive

Il campo delle scienze cognitive è davvero affascinante, e merita da solo numerosi libri e un'intera enciclopedia (Wilson e Keil, 1999). Occasionalmente ci potrà capitare di commentare i punti in comune e le differenze tra la cognizione umana e l'IA. La vera scienza cognitiva, comunque, è necessariamente basata sull'investigazione sperimentale di vere persone e animali; lasceremo questo argomento ad altri libri, mentre noi partiremo sempre dall'ipotesi che il lettore compia i suoi esperimenti su computer.

Nei primi tempi dell'IA si faceva spesso confusione tra approcci diversi: un autore avrebbe potuto argomentare che un algoritmo eseguiva efficacemente un'attività e *quindi* era un buon modello dell'esecuzione umana, o viceversa. Gli autori moderni separano chiaramente le due cose, e questa distinzione ha aiutato lo sviluppo sia dell'IA che delle scienze cognitive. I due campi si influenzano positivamente a vicenda, specialmente nell'area della visione artificiale, che incorpora risultati della sperimentazione neurofisiologica in modelli computazionali. Recentemente, la combinazione di metodi di neuroimaging e tecniche di apprendimento automatico per analizzare i dati ha portato a una prima capacità di "leggere menti", ovvero determinare il contenuto semantico degli intimi pensieri di una persona. Questa capacità potrebbe a sua volta gettare luce sul funzionamento della cognizione umana.

1.1.3 Pensare razionalmente: l'approccio delle "leggi del pensiero"

Il filosofo greco Aristotele è stato uno dei primi a cercare di codificare formalmente il "pensiero corretto", ovvero i processi di ragionamento irrefutabili. I suoi **sillogismi** forniscono schemi di deduzione che portano sempre a conclusioni corrette quando sono corrette le pre-

sillogismo

logica

messe. L'esempio canonico parte da “Socrate è un uomo” e “tutti gli uomini sono mortali” per arrivare a concludere che “Socrate è mortale” (questo esempio si deve probabilmente a Sesto Empirico e non ad Aristotele). Si riteneva che queste leggi del pensiero governassero il funzionamento della mente; il loro studio ha dato origine alla disciplina chiamata **logica**.

tradizione logicista

I logici del XIX secolo hanno sviluppato una notazione precisa per formulare enunciati riguardanti gli oggetti del mondo e le relazioni tra essi: tale notazione è molto più espressiva di quella aritmetica, che fornisce soltanto enunciati sui *numeri*. Già nel 1965 esistevano programmi che potevano, in linea di principio, risolvere qualsiasi problema descritto in linguaggio logico. La **tradizione logicista**, come viene chiamata all'interno dell'IA, spera di partire da programmi siffatti per costruire sistemi intelligenti.

probabilità

La logica, nella sua accezione convenzionale, richiede una conoscenza del mondo che sia *certa*, condizione che raramente si verifica in realtà. Per esempio, non conosciamo le regole della politica o della guerra nello stesso modo in cui conosciamo le regole degli scacchi o dell'aritmetica. La teoria delle **probabilità** va a colmare questa lacuna, consentendo un ragionamento rigoroso in presenza di informazioni incerte. In linea di principio ciò consente di costruire un modello del pensiero razionale che, partendo dalle informazioni percettive grezze, giunga a una comprensione di come funziona il mondo e alla capacità di prevedere il futuro. Ma non genera un *comportamento* intelligente; per questo abbiamo bisogno di una teoria dell'azione razionale. Il pensiero razionale, di per sé, non è sufficiente.

agente

1.1.4 Agire razionalmente: l'approccio degli agenti razionali

agente razionale

Un **agente** è semplicemente qualcosa che agisce, che fa qualcosa. Naturalmente tutti i programmi per computer fanno qualcosa, tuttavia si suppone che gli agenti artificiali facciano di più: operare autonomamente, essere in grado di percepire l'ambiente, persistere in un'attività per un lungo arco di tempo, adattarsi ai cambiamenti e creare e perseguire degli obiettivi. Un **agente razionale** agisce in modo da ottenere il miglior risultato o, in condizioni di incertezza, il miglior risultato atteso.

Nell'approccio all'IA basato sulle “leggi del pensiero”, l'enfasi è posta sulla correttezza delle inferenze. Essere in grado di formulare deduzioni corrette è talvolta parte di un agente razionale, perché un modo di agire razionalmente è dedurre che una data azione sia la migliore e agire quindi in tal senso. D'altra parte, ci sono anche tipologie di comportamento razionale che non coinvolgono l'inferenza logica: per esempio, ritirare la mano da una stufa rovente è un'azione di riflesso che porta solitamente a vantaggi maggiori di un'azione più lenta, compiuta dopo un attento ragionamento.

Tutte le abilità richieste dal test di Turing consentono a un agente di agire razionalmente. La rappresentazione della conoscenza e il ragionamento consentono agli agenti di prendere le giuste decisioni. Dobbiamo essere capaci di generare frasi comprensibili in linguaggio naturale, per vivere in una società complessa. L'apprendimento ci serve non solo per accrescere le nostre conoscenze, ma anche perché ci consente di migliorare la nostra capacità di generare un comportamento efficace, soprattutto quando ci troviamo in circostanze nuove.

L'approccio all'IA degli agenti razionali presenta due vantaggi rispetto agli altri. Prima di tutto, è più generale dell'approccio basato sulle “leggi del pensiero”, poiché il corretto uso dell'inferenza è solo uno di molteplici meccanismi utilizzabili per arrivare alla razionalità. In secondo luogo, si presta meglio a recepire gli sviluppi scientifici. La razionalità è ben definita matematicamente e si può considerare un aspetto generale. Spesso è possibile basarsi su questa specifica per ricavare progetti di agenti in grado di essere razionali in modo dimostrabile, mentre è generalmente impossibile dimostrare di aver raggiunto lo scopo di imitare il comportamento umano o i processi di pensiero.

Per questi motivi, l'approccio all'IA degli agenti razionali è risultato quasi sempre prevalente nella storia della disciplina. Nei primi decenni gli agenti razionali erano costruiti ba-

sandosi su fondamenti logici e con piani ben definiti per raggiungere scopi specifici. In seguito, metodi basati sulla teoria delle probabilità e sull'apprendimento automatico hanno consentito di creare agenti in grado di prendere decisioni in condizioni di incertezza per raggiungere il miglior risultato atteso. In sostanza, l'*IA si è concentrata sullo studio e la costruzione di agenti che fanno la cosa giusta*. Che cosa sia la cosa giusta dipende dall'obiettivo fornito all'agente. Questo paradigma generale è talmente pervasivo che potremmo chiamarlo **modello standard**. Prevale non solo nell'IA ma anche nella teoria del controllo, in cui un controllore minimizza una funzione di costo; nella ricerca operativa, in cui una politica massimizza una somma di ricompense; in statistica, dove una regola decisionale minimizza una funzione di costo; e in economia, dove un decisore massimizza l'utilità o qualche misura di benessere sociale.



fare la cosa giusta
modello standard

Dobbiamo apportare un importante aggiustamento al modello standard in modo da tenere conto del fatto che la razionalità perfetta – fare sempre l'azione perfettamente ottimale – non è praticabile in ambienti complessi, dati i requisiti di calcolo eccessivi. Nei Capitoli 5 e 17 affronteremo il tema della **razionalità limitata**, che consiste nell'agire in modo appropriato quando non vi è tempo sufficiente per eseguire tutti i calcoli che si vorrebbero svolgere. In ogni caso, la razionalità perfetta rimane un buon punto di partenza per l'analisi teorica.

razionalità limitata

1.1.5 Macchine che portano benefici

Il modello standard ha costituito un'utile guida per l'IA fin dalla sua origine, ma probabilmente non è un modello adatto per il lungo termine, poiché assume che si fornisca alla macchina un obiettivo specificato in modo completo.

Per un'attività definita in modo artificiale, come il gioco degli scacchi o il calcolo del cammino minimo, l'obiettivo è già integrato, per cui il modello standard risulta applicabile. Quando si passa al mondo reale, tuttavia, diviene sempre più difficile specificare l'obiettivo in modo completo e corretto. Per esempio, nel progettare un'automobile a guida autonoma, si potrebbe pensare che l'obiettivo sia quello di raggiungere la destinazione in modo sicuro. Ma percorrere le strade in auto comporta un rischio di incidenti dovuti a errori di altri guidatori, guasti dell'automobile e così via, perciò il requisito stretto della sicurezza richiederebbe di rimanere fermi in garage. Si ha quindi una situazione di compromesso tra procedere verso la destinazione e incorrere nel rischio di incidenti; come affrontarla? Inoltre, fino a quale punto possiamo consentire all'automobile di intraprendere azioni che potrebbero recare disagio ad altri guidatori? In che modo l'auto dovrebbe moderare accelerazioni, sterzate e frenate per evitare scossoni ai passeggeri? È difficile rispondere a priori a domande di questo tipo, che risultano particolarmente problematiche nell'area generale dell'interazione tra uomo e robot, in cui rientrano anche le automobili a guida autonoma.

Il problema di raggiungere un accordo tra le nostre reali preferenze e l'obiettivo posto nella macchina è detto **problema di allineamento dei valori**: i valori o obiettivi affidati alla macchina devono essere allineati a quelli dell'uomo. Se stiamo sviluppando un sistema di IA in laboratorio o in un simulatore, come da tradizione di questo campo, esiste un modo facile per rimediare a un obiettivo specificato in modo errato: resettare il sistema, correggere l'obiettivo e riprovare. Ma il campo dell'IA si dirige sempre di più verso sistemi intelligenti potenti e capaci che operano nel mondo reale, per cui questo approccio non è più praticabile. Un sistema messo all'opera con un obiettivo errato porterà a conseguenze negative, tanto più negative quanto più intelligente è il sistema.

**problema
di allineamento
dei valori**

Tornando all'esempio degli scacchi, apparentemente non problematico, pensiamo a che cosa accade se la macchina è tanto intelligente da ragionare e agire anche oltre i confini della scacchiera. In tal caso potrebbe tentare di aumentare le sue possibilità di vincere ricorrendo a stratagemmi come ipnotizzare o ricattare l'avversario, o convincendo il pubblico a rumo-



beneficio dimostrabile

reggiare per infastidire l'avversario mentre pensa alla prossima mossa.³ Potrebbe anche tentare di procurarsi illecitamente una maggiore potenza di calcolo. *Questi comportamenti non sono “stupidi” o “insani”; sono una conseguenza logica del definire la vittoria come unico e solo obiettivo della macchina.*

È impossibile prevedere tutti i modi in cui una macchina potrebbe adottare un comportamento errato nel perseguire un obiettivo fissato. Esiste quindi una buona ragione per pensare che il modello standard sia inadeguato. Non vogliamo macchine intelligenti nel senso che perseguono i *loro* obiettivi, vogliamo che perseguano i *nostri* obiettivi. Se non riusciamo a trasferire i nostri obiettivi alla macchina in modo perfetto, ci serve una nuova formulazione in cui la macchina persegue i nostri obiettivi, ma è necessariamente *incerta* su quali siano. Quando una macchina sa di non conoscere l'obiettivo completo, ha un incentivo ad agire in modo cauto, a chiedere il permesso, a capire meglio le nostre preferenze attraverso l'osservazione, a rimettersi al controllo dell'uomo. In definitiva, vogliamo avere agenti che portino **benefici dimostrabili** agli esseri umani. Torneremo su questo argomento nel Paragrafo 1.5.

1.2 I fondamenti dell'intelligenza artificiale

In questo paragrafo forniamo una breve panoramica delle discipline che hanno contribuito all'IA con idee, punti di vista e tecniche. Come tutte le trattazioni storiche, anche questa si concentra su un piccolo numero di persone, eventi e idee, ignorandone altre ugualmente importanti. La nostra esposizione è organizzata intorno a un piccolo insieme di quesiti: non vogliamo certo intendere che queste siano le uniche domande poste da tali discipline, o che esse abbiano avuto come scopo ultimo quello di contribuire all'IA.

1.2.1 Filosofia

- È possibile applicare regole formali per trarre conclusioni valide?
- In che modo la mente scaturisce dal cervello fisico?
- Da dove proviene la conoscenza?
- Come fa la conoscenza a trasformarsi in azione?

Aristotele (384–322 a.C.) fu il primo filosofo a formulare un insieme preciso di leggi che governano la parte razionale della mente. Egli sviluppò un sistema informale di sillogismi per il ragionamento corretto, che in via di principio consentivano a chiunque, date le premesse iniziali, di generare meccanicamente le conclusioni.

Raimondo Lullo (1232–1315) ideò un sistema di ragionamento che descrisse in *Ars Magna* (*La grande arte*) e tentò di implementarlo utilizzando un dispositivo meccanico costituito da una serie di ruote di carta che si potevano girare in diverse permutazioni.

Intorno al 1500 Leonardo da Vinci (1452–1519) progettò, ma non costruì, un calcolatore meccanico; alcune ricostruzioni recenti hanno dimostrato che il progetto era corretto. La prima macchina calcolatrice fu costruita attorno al 1623 dallo scienziato tedesco Wilhelm Schickard (1592–1635). Blaise Pascal (1623–1662) costruì nel 1642 la Pascalina e scrisse che tale macchina aritmetica “produce effetti che sembrano più vicini al pensiero di tutte le azioni degli animali”. Gottfried Wilhelm Leibniz (1646–1716) costruì un dispositivo meccanico concepito per eseguire operazioni su concetti invece che numeri, ma le sue capacità erano

³ In uno dei primi libri sugli scacchi, Ruy Lopez (1561) scrisse: “Posiziona sempre la scacchiera in modo che il tuo avversario abbia il sole negli occhi”.

piuttosto limitate. Nel libro *Leviathan* (trad. it. *Leviatano*) del 1651 Thomas Hobbes (1588–1679) suggerì l'idea di una macchina pensante, un “animale artificiale” per usare le sue parole, argomentando: “Che cos’è il cuore se non una molla; e i nervi, se non tante cinghie; e le articolazioni, se non tante ruote”. Suggerì inoltre che il ragionamento fosse come il calcolo numerico: “Perché ‘ragionare’ non è altro che ‘calcolare’, cioè sommare e sottrarre”.

Un conto è dire che la mente opera almeno in parte secondo regole logiche, e costruire sistemi fisici che emulino alcune di tali regole, altro è dire che la mente stessa è un sistema fisico di tal tipo. Cartesio (1596–1650) avviò la prima discussione chiara sulla distinzione tra mente e materia. Egli notò che una concezione puramente fisica della mente sembra lasciare poco spazio al libero arbitrio: se il cervello è governato interamente da leggi fisiche, allora non ha più volontà di una roccia che “decide” di cadere verso il basso. Cartesio era un sostenitore del **dualismo**, secondo il quale esisteva una parte della mente umana (anima o spirito) esente dalle leggi fisiche, al di fuori della natura. Gli animali, al contrario, non possedevano questa natura duale, e si sarebbero potuti trattare come macchine.

Un’alternativa al dualismo è il **materialismo**, che sostiene che il funzionamento del cervello secondo le leggi della fisica *costituisce* la mente. Il libero arbitrio è semplicemente il modo in cui la percezione delle scelte disponibili si manifesta all’agente che deve scegliere. Questa visione, che contrasta con il sovrannaturale, è descritta anche con i termini **fiscalismo** e **naturalismo**.

Data una mente fisica che manipola la conoscenza, il problema successivo è stabilire la fonte di tale conoscenza. Il movimento dell'**empirismo**, che prende piede dal *Novum Organum* di Francis Bacon (1561–1626),⁴ è caratterizzato dal detto di John Locke (1632–1704): “Non c’è nulla nell’intelletto, che non sia stato prima nei sensi”.

David Hume (1711–1776) in *A treatise of human nature* (1739; trad. it. *Trattato sulla natura umana*) propose quello che oggi è noto come principio di **induzione**: le regole generali sono inferite da ripetute associazioni tra i loro elementi.

Sviluppando il lavoro di Ludwig Wittgenstein (1889–1951) e Bertrand Russell (1872–1970), il famoso Circolo di Vienna (Sigmund, 2017), un gruppo di filosofi e matematici che si incontravano a Vienna negli anni 1920 e 1930, sviluppò la dottrina del **positivismo logico**. Questa teoria sostiene che tutta la conoscenza può essere espressa da teorie collegate, alla fine, a **enunciati osservativi** che corrispondono alle percezioni sensoriali; perciò il positivismo logico combina razionalismo ed empirismo.

La **teoria della conferma** di Rudolf Carnap (1891–1970) e Carl Hempel (1905–1997) cercò di analizzare l’acquisizione di conoscenza dall’esperienza quantificando il grado di fiducia da assegnare alle proposizioni logiche in base alla loro connessione a osservazioni che le confermavano o smentivano. Il libro di Carnap *Der Logische Aufbau der Welt* (1928; trad. it. *La costruzione logica del mondo*) fu forse il primo a presentare una teoria della mente come processo computazionale.

L’ultimo elemento della concezione filosofica della mente è il collegamento tra conoscenza e azione. Questo aspetto è fondamentale per l’IA, perché l’intelligenza richiede tanto il ragionamento quanto l’azione. Inoltre, solo comprendendo le ragioni dietro le azioni possiamo capire come costruire un agente le cui azioni siano giustificabili (o razionali).

Aristotele sostenne (in *De Motu Animalium*) che le azioni sono giustificate da un collegamento logico tra gli scopi dell’agente e la conoscenza del risultato di ogni azione:

Ma perché succede che certe volte il pensiero sia accompagnato dall’azione e altre volte no, certe volte dal moto e altre volte no? Sembra che accada quasi la stessa cosa che si verifica nel caso del ragionamento e delle inferenze riguardo oggetti immutabili. Ma in quel caso il risultato è una

dualismo

empirismo

induzione

positivismo logico

**enunciato
osservativo**

**teoria della
conferma**

⁴ Il *Novum Organum* è un aggiornamento dell’*Organon* di Aristotele, inteso come strumento del pensiero.

proposizione speculativa (...) laddove qui la conclusione che scaturisce da due premesse è un’azione. (...) Ho bisogno di coprirmi; un mantello offre copertura. Ho bisogno di un mantello. Quello di cui ho bisogno, devo farlo; ho bisogno di un mantello. Devo farmi un mantello. E la conclusione, “devo farmi un mantello”, è un’azione.

Nell’*Etica Nicomachea* (Libro III. 3, 1112b), Aristotele sviluppa ulteriormente questo argomento, suggerendo un algoritmo:

Le nostre decisioni non riguardano i fini, ma i mezzi. Infatti un medico non decide se deve curare, né un oratore se deve convincere, (...) Essi danno per scontato il fine e considerano come e con quali mezzi ottenerlo; e se sembra possibile raggiungerlo in più modi considerano quale sia il più facile ed efficace, mentre se il mezzo è uno solo allora considerano *come* raggiungere il fine mediante quello e quali mezzi usare per raggiungere *quello*, e così via finché non raggiungono la causa prima, (...) e ciò che viene ultimo in ordine di analisi sembra essere la prima cosa a essere messa in pratica. E se incontriamo un passaggio irrisolvibile dobbiamo abbandonare la ricerca, per esempio se abbiamo bisogno di denaro e non possiamo procurarcelo; ma se una cosa sembra possibile allora cerchiamo di farla.

L’algoritmo di Aristotele è stato implementato 2300 anni dopo da Newell e Simon nel loro programma **General Problem Solver**. Ora lo definiremmo un sistema di pianificazione greedy con regressione (cfr. Capitolo 11). Nei primi decenni, la ricerca teorica nel campo dell’IA fu dominata da metodi che si basavano sulla pianificazione logica per raggiungere obiettivi definiti.

Ragionare esclusivamente in termini di azioni che raggiungono obiettivi è spesso utile, ma a volte inapplicabile. Per esempio, se esistono diversi modi per raggiungere un obiettivo, deve esistere un modo per effettuare una scelta fra di essi. Cosa ancora più importante, a volte è impossibile raggiungere un obiettivo con certezza, ma un’azione va comunque intrapresa. Come si fa a decidere? Antoine Arnauld (1662), analizzando il concetto di decisioni razionali nel gioco d’azzardo, propose una formula quantitativa per massimizzare il valore monetario atteso del risultato. In seguito, Daniel Bernoulli (1738) introdusse il concetto più generale di **utilità** per rappresentare il valore soggettivo di un risultato. Il concetto moderno di prendere decisioni razionali in condizioni di incertezza comporta la massimizzazione dell’utilità attesa, come viene spiegato nel Capitolo 16.

utilità

In tema di etica e politica pubblica, un decisore deve considerare gli interessi di una molteplicità di individui. Jeremy Bentham (1823) e John Stuart Mill (1863) promossero il concetto di **utilitarismo**, secondo cui il processo di prendere decisioni razionali in base alla massimizzazione dell’utilità attesa doveva applicarsi a tutte le sfere dell’attività umana, incluse le decisioni di politica pubblica prese per conto di moltissimi individui. L’utilitarismo è un tipo particolare di **consequenzialismo**: l’idea che ciò che è giusto o sbagliato sia determinato dai risultati attesi di un’azione.

utilitarismo

Immanuel Kant, invece, nel 1785 propose una teoria della **deontologia** o etica basata sulle regole, in cui “la cosa giusta da fare” è determinata non dai risultati, ma dalle leggi sociali universali che governano le azioni lecite, come “non mentire” o “non uccidere”. Quindi, un utilitarista potrebbe raccontare una bugia innocua se il beneficio atteso supera gli svantaggi, ma un kantiano non potrebbe farlo, perché mentire è sempre sbagliato. Mill riconosceva il valore delle regole, ma le interpretava come efficienti procedure decisionali elaborate mediante un ragionamento sulle conseguenze che partiva dai principi primi. Molti sistemi di IA moderni adottano esattamente questo approccio.

deontologia

1.2.2 Matematica

- Quali sono le regole formali utilizzabili per trarre conclusioni valide?
- Cosa può essere calcolato?
- Come si deve ragionare quando l’informazione è incerta?

I filosofi hanno studiato la maggior parte dei concetti fondamentali riguardanti l'IA, ma il passaggio a una scienza formale richiedeva la matematizzazione di logica e probabilità e l'introduzione di un nuovo ramo della matematica: la computazione.

L'idea di una **logica formale** si può far risalire ai filosofi dell'antica Grecia, dell'India e della Cina, ma il suo sviluppo matematico cominciò effettivamente con il lavoro di George Boole (1815–1864), che formulò i principi della logica proposizionale o booleana (Boole, 1847). Nel 1879, Gottlob Frege (1848–1925) estese la logica di Boole in modo da includere oggetti e relazioni, creando così la logica del primo ordine usata oggi.⁵ Oltre ad avere un ruolo centrale nel primo periodo delle ricerche in IA, la logica del primo ordine stimolò i lavori di Gödel e Turing che andarono a costituire le basi della computazione stessa, come spieghiamo nel seguito.

La teoria delle **probabilità** può essere vista come una generalizzazione della logica a situazioni con informazioni incerte, considerazione di grande importanza per l'IA. Gerolamo Cardano (1501–1576) fu il primo a definire il concetto di probabilità, descrivendolo in termini dei risultati possibili di eventi del gioco d'azzardo. Nel 1654 Blaise Pascal (1623–1662), in una lettera a Pierre Fermat (1601–1665), mostrò come predire il futuro di un gioco non ancora concluso e assegnare premi medi ai giocatori. La probabilità divenne presto una parte fondamentale delle scienze quantitative, rivelandosi utile per affrontare misure incerte e teorie incomplete. Jacob Bernoulli (1654–1705, zio di Daniel), Pierre Laplace (1749–1827) e altri fecero avanzare la teoria e introdussero nuovi metodi statistici. Thomas Bayes (1702–1761) propose una regola per aggiornare le probabilità alla luce di nuove evidenze; la regola di Bayes è uno strumento fondamentale per i sistemi di IA. La formalizzazione del concetto di probabilità, insieme alla disponibilità dei dati, portò all'emergere del campo della **statistica**. Una delle prime applicazioni fu l'analisi svolta da John Graunt dei dati censuari di Londra nel 1662. Ronald Fisher, considerato il primo statistico moderno (Fisher, 1922), mise insieme i concetti di probabilità, progettazione degli esperimenti, analisi dei dati e computazione; nel 1919 affermò con forza che non avrebbe potuto svolgere il suo lavoro senza un calcolatore meccanico chiamato MILLIONNAIRE (il primo calcolatore in grado di eseguire moltiplicazioni), anche se il costo del calcolatore era superiore al suo salario di un anno (Ross, 2012).

La storia della computazione è antica quanto la storia dei numeri, ma si ritiene che il primo **algoritmo** non banale sia quello di Euclide per calcolare il massimo comune denominatore. Il termine *algoritmo* risale a Muhammad ibn Musa al-Khwarizmi, un matematico del IX secolo i cui lavori introdussero in Europa anche l'uso dei numeri arabi e dell'algebra. Boole e altri considerarono lo sviluppo di algoritmi per la deduzione logica, e verso la fine del XIX secolo molti scienziati dedicarono la loro ricerca alla formalizzazione di ragionamenti matematici generali sotto forma di deduzione logica.

Kurt Gödel (1906–1978) mostrò che esiste una procedura di calcolo per dimostrare ogni enunciato vero nella logica del primo ordine di Frege e Russell, ma che tale logica non può esprimere il principio di induzione matematica necessario per definire i numeri naturali. Nel 1931 Gödel dimostrò che esistono dei limiti per la deduzione: il suo **teorema di incompletezza** stabilisce che all'interno di qualsiasi teoria formale di potenza almeno pari a quella dell'aritmetica di Peano (la teoria elementare dei numeri naturali) esistono enunciati necessariamente veri che non sono dimostrabili rimanendo all'interno della teoria.

Questo risultato fondamentale si può interpretare anche come la dimostrazione che esistono funzioni sui numeri interi che non possono essere rappresentate per via algoritmica, e sono quindi incomputabili. Questo spinse Alan Turing (1912–1954) a cercare di definire esattamente quali funzioni sono **computabili**, cioè calcolabili con una procedura di calcolo.

logica formale

probabilità

statistica

algoritmo

teorema di incompletezza

computabile

⁵ La notazione proposta da Frege per la logica del primo ordine – un'arcana combinazione di caratteristiche testuali e geometriche – non ha mai riscosso grande successo.

trattabilità

La tesi di Church–Turing propone di identificare il concetto generale di computabilità con le funzioni calcolate da una macchina di Turing (Turing, 1936). Turing ha anche dimostrato che esistono funzioni che nessuna macchina di Turing può calcolare. Per esempio, nessuna macchina può dire se *in generale* un dato programma, ricevuto un determinato input, restituirà un risultato o continuerà l'esecuzione per sempre.

NP-completezza

Benché il concetto di computabilità sia importante per comprendere i principî del calcolo automatico, quello di **trattabilità** ha avuto un impatto ancora superiore sull'IA. Semplificando, si può dire che un problema è intrattabile quando il tempo richiesto per risolvere una sua determinata istanza cresce esponenzialmente con la dimensione dell'istanza stessa. La distinzione tra crescita polinomiale ed esponenziale nella complessità degli algoritmi fu enfatizzata per la prima volta a metà degli anni 1960 (Cobham, 1964; Edmonds, 1965). Si tratta di un concetto fondamentale, perché una crescita esponenziale significa che istanze del problema anche moderatamente complesse non possono essere risolte in tempi ragionevoli.

La teoria della **NP-completezza**, formulata inizialmente da Cook (1971) e Karp (1972), fornisce una base per analizzare la trattabilità dei problemi; ogni classe di problemi a cui si può ridurre quella dei problemi NP-completi è probabilmente intrattabile: in effetti non è stato dimostrato che i problemi NP-completi siano necessariamente intrattabili, ma la maggior parte degli esperti ritiene che sia così. Questi risultati contrastano con l'ottimismo con cui la stampa non specializzata aveva salutato i primi computer – “supercervelloni elettronici” che erano “più veloci di Einstein”! Nonostante la velocità sempre maggiore dei calcolatori, i sistemi intelligenti dovranno sempre fare un uso molto accurato delle risorse computazionali disponibili. Per dirla in parole povere, il mondo è un'istanza di problema *davvero grande!*

1.2.3 Economia

- Come dobbiamo prendere decisioni in modo da seguire le nostre preferenze?
- Come dobbiamo farlo quando gli altri non sono d'accordo con noi?
- Come dobbiamo farlo quando i vantaggi potrebbero essere molto lontani nel futuro?

L'economia come scienza ebbe origine nel 1776, quando Adam Smith (1723–1790) pubblicò *An Inquiry into the Nature and Causes of the Wealth of Nations* (trad. it. *La ricchezza delle nazioni*). Smith propose di considerare le economie come se fossero costituite da molti agenti individuali che perseguono i propri interessi. Tuttavia, Smith non propugnava l'avidità finanziaria come posizione morale: il suo precedente (1759) libro *The Theory of Moral Sentiments* (trad. it. *Teoria dei sentimenti morali*) inizia evidenziando che la preoccupazione per il benessere altrui è un elemento essenziale degli interessi di ogni individuo.

La maggior parte delle persone ritiene che l'economia riguardi solo il denaro, e in effetti la prima analisi matematica delle decisioni in condizioni di incertezza, la formula del massimo valore atteso di Arnauld (1662), si occupava del valore monetario delle scommesse. Daniel Bernoulli (1738) osservò che tale formula non sembrava funzionare bene per quantità di denaro più elevate, come gli investimenti nelle spedizioni commerciali marittime; propose quindi un principio basato sulla massimizzazione dell'utilità attesa, e spiegò le scelte di investimento degli esseri umani suggerendo che l'utilità marginale di una data quantità aggiuntiva di denaro diminuisse all'aumentare del denaro acquisito da un individuo.

Émile Durkheim (1858–1917) fornì alla teoria dell'utilità un fondamento più generale in termini di preferenze tra scommesse su qualsiasi risultato (non solo monetario). La teoria fu migliorata da Ramsey (1931) e poi da John von Neumann e Oskar Morgenstern nel loro libro *The Theory of Games and Economic Behavior* (1944). L'economia non è più lo studio del denaro, ma lo studio di desideri e preferenze.

La **teoria delle decisioni**, che fonde la teoria delle probabilità con quella dell'utilità, fornisce un'infrastruttura formale completa in supporto alle decisioni (economiche e non) prese

teoria delle decisioni

in condizioni di incertezza, ovvero nei casi in cui l'ambiente decisionale può essere descritto in modo probabilistico. Ciò si adatta bene a economie “grandi” in cui ogni agente non ha bisogno di prestare attenzione alle azioni degli altri agenti o individui. Nelle economie “piccole”, la situazione assomiglia di più a un **gioco**: le azioni di un giocatore possono influenzare in modo significativo l'utilità di un altro, sia positivamente che negativamente. Lo sviluppo della **teoria dei giochi** da parte di von Neumann e Morgenstern (cfr. anche Luce e Raiffa, 1957) portò tra l'altro al risultato sorprendente secondo cui, in alcuni giochi, un agente razionale deve adottare politiche che siano (o almeno appaiano) casuali. A differenza della teoria delle decisioni, la teoria dei giochi non offre una prescrizione priva di ambiguità per la scelta di azioni. Nell'IA, le decisioni che coinvolgono una molteplicità di agenti sono studiate nell'ambito dei **sistemi multiagente** (Capitolo 18).

Gli economisti, con alcune eccezioni, non affrontarono la terza delle questioni elencate precedentemente, ovvero come prendere decisioni razionali quando gli effetti delle azioni non sono immediati, ma risultano invece dall'esecuzione di molteplici azioni *in sequenza*. Questo argomento fu affrontato nel campo della **ricerca operativa**, che scaturì durante la Seconda Guerra Mondiale dagli sforzi britannici per ottimizzare le installazioni radar, e più tardi trovò innumerevoli applicazioni in ambito civile. Gli studi di Richard Bellman (1957) portarono alla formalizzazione di una classe di problemi basati su sequenze di decisioni, chiamati **processi decisionali di Markov**, che esamineremo nel Capitolo 17 e, nell'ambito dell'**apprendimento per rinforzo** (*reinforcement learning*), nel Capitolo 22 (Volume 2).

ricerca operativa

Il lavoro svolto nei campi dell'economia e della ricerca operativa ha dato un grande contributo alla nostra nozione di agente intelligente; tuttavia, per molti anni l'IA ha sviluppato la propria ricerca lungo un cammino completamente separato. Una delle ragioni era l'apparente complessità intrinseca nell'attività di prendere decisioni razionali. Herbert Simon (1916–2001), uno dei pionieri dell'IA, vinse il premio Nobel per l'economia nel 1978 per il suo lavoro che mostrava che modelli basati sulla **soddisfazione** – in grado di prendere decisioni “abbastanza buone”, invece di calcolare faticosamente la decisione ottima – fornivano una descrizione più accurata dell'effettivo comportamento umano (Simon, 1947). A partire dagli anni 1990 c'è stato un ritorno di interesse per le tecniche della teoria delle decisioni applicate all'IA.

soddisfazione

1.2.4 Neuroscienze

- Come avviene l'elaborazione dell'informazione da parte del cervello?

Le **neuroscienze** si occupano dello studio del sistema nervoso, e in particolare del cervello. Benché il modo preciso in cui ha origine il pensiero rimanga uno dei grandi misteri della scienza, il ruolo del cervello nel rendere possibile la formazione del pensiero è stato postulato per migliaia di anni, in base al fatto che i colpi alla testa possono portare a deficit mentali. Anche il fatto che il cervello umano sia in qualche modo differente era risaputo; intorno al 335 a.C. Aristotele scrisse: “Di tutti gli animali, l'uomo ha il cervello più grande in proporzione alla sua massa”.⁶ Tuttavia, il cervello non fu riconosciuto comunemente come sede della coscienza fino alla metà del XVIII secolo: prima di allora, le ipotesi includevano tra l'altro il cuore e la milza.

neuroscienze

Nel 1861, le ricerche di Paul Broca (1824–1880) sulle afasie (deficit linguistici) nei pazienti con danni cerebrali iniziarono lo studio della struttura funzionale del cervello individuando un'area nell'emisfero sinistro, ora chiamata appunto area di Broca, che è responsabile della produzione del linguaggio.⁷ A quell'epoca si sapeva già che il cervello è formato da cellule

⁶ Da allora è stato scoperto che la tupaia (*Scadentia*) e alcune specie di uccelli hanno un rapporto più elevato tra massa cerebrale e massa corporea.

⁷ Molti citano Alexander Hood (1824) come possibile fonte precedente.

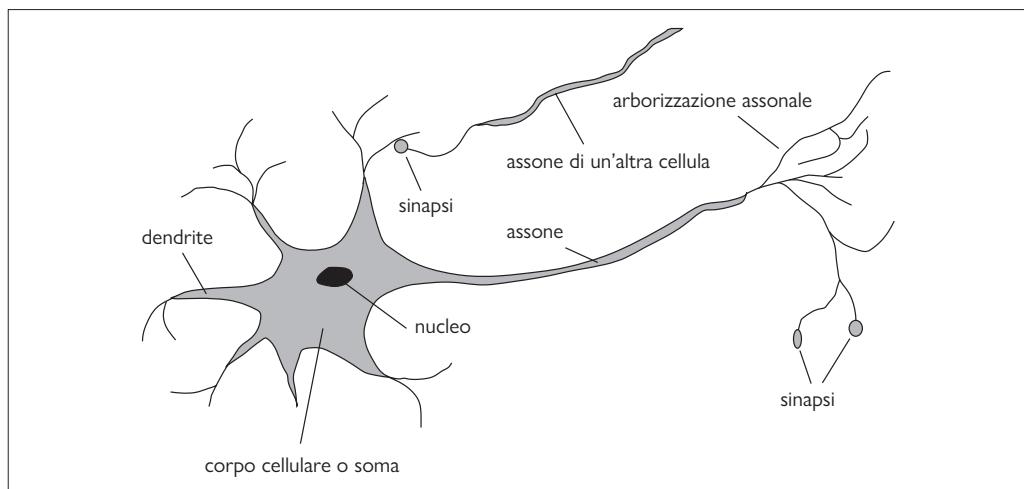


Figura 1.1 Le parti che compongono una cellula nervosa o neurone. Ogni neurone consiste in un corpo cellulare, o soma, che contiene il nucleo. Dal corpo si dirama una quantità di fibre chiamate dendriti e una singola, lunga fibra che prende il nome di assone. L'assone si prolunga per una grande distanza, maggiore di quella raffigurata nel diagramma: tipicamente gli assoni sono lunghi 1 cm (100 volte il diametro del corpo cellulare), ma possono arrivare fino a un metro. Un neurone si collega con un numero che va da 10 a 100.000 altri neuroni, utilizzando punti di congiunzione chiamati sinapsi. I segnali si propagano da un neurone all'altro grazie a una complicata reazione elettrochimica e controllano l'attività cerebrale nel breve periodo, ma permettono anche dei cambiamenti a lungo termine nelle connessioni tra i neuroni. Si ritiene che questo meccanismo formi la base dell'apprendimento. La maggior parte dell'elaborazione delle informazioni ha luogo nella corteccia, il rivestimento esterno del cervello. L'unità base di organizzazione sembra essere una colonna di tessuto di circa mezzo millimetro di diametro, che contiene circa 20.000 neuroni e si estende per tutta la profondità della corteccia, che negli esseri umani è di circa 4 mm.

neurone



nervo^e o **neuroni**, ma si dovette aspettare il 1873 affinché Camillo Golgi (1843–1926) svilupasse una tecnica che permettesse la visualizzazione di neuroni singoli (Figura 1.1). Questa tecnica fu usata da Santiago Ramon y Cajal (1852–1934) nei suoi pionieristici studi sulle strutture neuronali.⁸ Oggi vi è ampio consenso sul fatto che le funzioni cognitive derivino dall'attività elettrochimica di queste strutture, cioè che *un insieme di semplici cellule può creare pensiero, azione e coscienza*. Nelle parole di John Searle (1992), *il cervello è la causa della mente*.

Oggi possediamo alcuni dati relativi alla corrispondenza tra le aree del cervello e le parti del corpo da esse controllate, o da cui ricevono input sensoriali. Tali corrispondenze possono cambiare radicalmente nel corso di poche settimane, e alcuni animali sembrano avere corrispondenze multiple. Inoltre, non abbiamo ancora compreso appieno come opera il meccanismo in base al quale altre aree possono prendere il controllo di alcune funzioni quando si verificano dei danneggiamenti al tessuto nervoso. Praticamente non esiste alcuna teoria sulla memorizzazione dei singoli ricordi o sul meccanismo di funzionamento delle funzioni cognitive di livello più alto.

Le misurazioni dell'attività di un cervello integro cominciarono nel 1929 con l'invenzione dell'elettroencefalografo (EEG) da parte di Hans Berger. Lo sviluppo della risonanza magnetica funzionale (fMRI) (Ogawa *et al.*, 1990; Cabeza e Nyberg, 2001) sta fornendo ai neuroscienziati immagini dell'attività cerebrale con un dettaglio precedentemente impensabile, permettendo misurazioni che corrispondono in modo rilevante ai processi cognitivi in corso.

⁸ Golgi continuò a ritenere che il funzionamento del cervello avesse luogo principalmente all'interno di un mezzo continuo in cui erano inseriti i neuroni, mentre Cajal propendeva per la “dottrina neuronale”. I due si divisero il premio Nobel nel 1906, ma i rispettivi discorsi di accettazione furono alquanto critici l'uno dell'altro.

	Supercomputer	Personal Computer	Cervello umano
Unità di calcolo	10^6 GPU + CPU 10^{15} transistor	8 core CPU 10^{10} transistor	10^6 colonne 10^{11} neuroni
Unità di memorizzazione	10^{16} byte di RAM 10^{17} byte su disco	10^{10} byte di RAM 10^{12} byte su disco	10^{11} neuroni 10^{14} sinapsi
Tempo di ciclo	10^{-9} secondi	10^{-9} secondi	10^{-3} secondi
Operazioni/secondo	10^{18}	10^{10}	10^{17}

Figura 1.2 Un confronto approssimativo tra un importante supercomputer, Summit (Feldman, 2017), un tipico personal computer del 2019 e il cervello umano. La potenza del cervello umano non è cambiata molto in migliaia di anni, mentre i supercomputer sono passati dai megaFLOP degli anni 1960 ai gigaFLOP degli anni 1980, ai teraFLOP degli anni 1990, ai petaFLOP del 2008 e agli exaFLOP del 2018 (1 exaFLOP = 10^{18} operazioni in virgola mobile al secondo).

Questi nuovi dati sono poi arricchiti dai progressi che consentono di registrare l'attività elettrica dei neuroni e dai metodi dell'**optogenetica** (Crick, 1999; Zemelman *et al.*, 2002; Han e Boyden, 2007) che consentono la misura e il controllo di singoli neuroni modificati in modo da risultare sensibili alla luce.

optogenetica

Lo sviluppo di **interfacce cervello–macchina** (Lebedev e Nicolelis, 2006) per percezione sensoriale e controllo motorio non solo promette di poter ripristinare alcune funzioni nelle persone disabili, ma getta nuova luce su molti aspetti dei sistemi neurali. Una notevole scoperta ottenuta in quest'ambito è che il cervello è in grado di modificarsi per interfacciarsi con un dispositivo esterno, considerando quest'ultimo come un altro organo sensoriale o arto.

interfaccia
cervello–macchina

I cervelli e i computer digitali hanno caratteristiche differenti. La Figura 1.2 mostra che i computer hanno un tempo di ciclo un milione di volte più rapido rispetto al cervello. Il cervello compensa con una quantità molto maggiore di spazio di memorizzazione e interconnessione rispetto a un personal computer ad alte prestazioni, anche se i più grandi supercomputer hanno capacità simile a quella del cervello umano secondo alcune metriche. I futuologi si basano pesantemente su questi numeri per indicare che stiamo avvicinandoci a una **singolarità** in cui i computer raggiungeranno un livello di prestazioni superiore a quello dell'uomo (Vinge, 1993; Kurzweil, 2005; Doctorow e Stross, 2012), ma i semplici confronti fra numeri grezzi non risultano particolarmente informativi. Anche disponendo di un computer di capacità virtualmente illimitata, ci servono ulteriori progressi concettuali nella nostra comprensione dell'intelligenza (cfr. Capitolo 28). Per dirla in parole semplici: senza la teoria giusta, macchine più veloci forniscono solo risposte sbagliate più rapidamente.

singolarità

1.2.5 Psicologia

- Come pensano e agiscono gli esseri umani e gli animali?

Le origini della psicologia come scienza vengono solitamente fatte risalire al lavoro del medico tedesco Hermann von Helmholtz (1821–1894) e del suo studente Wilhelm Wundt (1832–1920). Helmholtz applicò il metodo scientifico allo studio della visione umana, e il suo *Lehrbuch der Physiologie des Menschen* è stato definito “il singolo trattato più importante sulla fisica e la fisiologia della visione umana” (Nalwa, 1993, p.15). Nel 1879, Wundt aprì il primo laboratorio di psicologia sperimentale presso l’Università di Lipsia ed effettuò esperimenti strettamente controllati, nei quali i suoi collaboratori dovevano eseguire qualche attività percettiva o associativa mentre si concentravano introspettivamente sui loro processi mentali. I controlli accurati contribuirono molto alla trasformazione della psicologia in una scienza, ma data la natura soggettiva dei dati, era improbabile che l'autore di un esperimento riconoscesse la falsità delle proprie teorie.

movimento behaviorista**psicologia cognitiva****intelligenza aumentata**

I biologi che studiavano il comportamento animale, al contrario, non potevano basarsi su dati soggettivi e dovettero sviluppare una metodologia oggettiva, come descritto da H. S. Jennings (1906) nel suo influente lavoro *The Behavior of the Lower Organisms*. Applicando agli esseri umani questo punto di vista, il **movimento behaviorista**, guidato da John Watson (1878–1958), rifiutò *qualsiasi* teoria riguardante i processi mentali sulla base dell'affermazione che l'introspezione non poteva fornire alcun dato affidabile. I behavioristi insistevano sullo studio esclusivo delle misurazioni delle percezioni (o *stimoli*) forniti a un animale e delle risultanti azioni (o *risposte*). Il behaviorismo riuscì a scoprire parecchie cose sui ratti e i piccioni, ma ebbe meno fortuna nella comprensione degli esseri umani.

La **psicologia cognitiva**, in cui il cervello è visto come un dispositivo per l'elaborazione di informazioni, si può far risalire almeno ai lavori di William James (1842–1910). Anche Helmholtz sostenne che la percezione implicasse una qualche forma di inferenza logica inconscia. Negli Stati Uniti il punto di vista cognitivo fu in gran parte eclissato dal behaviorismo, ma alla Applied Psychology Unit di Cambridge, diretta da Frederic Bartlett (1886–1969), la modellazione cognitiva poté invece fiorire. *The Nature of Explanation* (1943), scritto da uno studente e successore di Bartlett di nome Kenneth Craik, ristabilì con forza la legittimità di termini "mentali" come credenza e scopo, sostenendo che fossero altrettanto scientifici di termini come pressione e temperatura quando si parla di gas, benché i gas siano fatti di molecole per le quali non ha senso parlare di tali grandezze.

Craik specificò tre requisiti fondamentali per un agente basato sulla conoscenza: (1) lo stimolo dev'essere tradotto in una rappresentazione interna; (2) la rappresentazione dev'essere manipolata da processi cognitivi per ottenere nuove rappresentazioni interne; (3) queste ultime devono essere a loro volta trasformate in azioni. La sua spiegazione del perché questo sia un buon progetto per un agente è chiara:

Se l'organismo porta nella sua testa un "modello in scala" della realtà esterna e delle proprie possibili azioni sarà in grado di provare varie alternative, decidere quali di esse sia la migliore, reagire a situazioni future prima che si manifestino, utilizzare la conoscenza di eventi passati per gestire quelli presenti e futuri, e sotto ogni aspetto reagire in modo molto più ricco, affidabile e competente alle emergenze che si troverà a fronteggiare. (Craik, 1943)

Dopo la morte di Craik per un incidente di bicicletta nel 1945, il suo lavoro fu portato avanti da Donald Broadbent, il cui libro *Perception and Communication* (1958) fu tra i primi lavori a modellare i fenomeni psicologici basandosi sull'elaborazione di informazioni. Intanto, negli Stati Uniti, lo sviluppo di modelli basati su computer portò alla creazione del nuovo campo della **scienza cognitiva**. Si può dire che la disciplina sia nata in durante un workshop nel settembre del 1956 al MIT – soltanto due mesi dopo quello che a Dartmouth aveva segnato la "nascita" dell'IA. A quel workshop George Miller presentò *The Magic Number Seven*, Noam Chomsky *Three Models of Language*, Allen Newell e Herbert Simon *The Logic Theory Machine*. Questi tre lavori fondamentali mostrarono come i modelli basati su computer potevano essere usati per affrontare rispettivamente la psicologia della memoria, del linguaggio e del pensiero logico. Oggi è una considerazione comune (benché non certo universale) tra gli psicologi che "una teoria cognitiva dovrebbe essere come un programma per computer" (Anderson, 1980), ovvero dovrebbe descrivere il meccanismo operativo di una funzione cognitiva in termini di elaborazione di informazioni.

Per i nostri scopi, includeremo il campo dell'**interazione uomo-computer** (HCI, *human-computer interaction*) nell'ambito della psicologia. Doug Engelbart, uno dei pionieri dell'interazione uomo-computer, promosse il concetto di **intelligenza aumentata**, sostenendo che i computer dovessero aumentare le capacità umane anziché automatizzare le attività dell'uomo. Nel 1968 Engelbart con la "madre di tutte le dimostrazioni" presentò per la prima volta il mouse, un sistema a finestre, l'ipertesto e la videoconferenza, nel tentativo di mostrare ciò che gli umani lavoratori della conoscenza erano in grado di ottenere insieme grazie all'intelligenza aumentata.

Oggi tendiamo a considerare l'intelligenza aumentata e l'intelligenza artificiale come due facce della stessa medaglia: la prima pone l'enfasi sul controllo umano mentre la seconda enfatizza il comportamento intelligente da parte della macchina. Entrambe sono necessarie affinché le macchine possano risultare utili all'uomo.

1.2.6 Ingegneria informatica

- Com'è possibile costruire un computer efficiente?

Il moderno computer elettronico digitale fu inventato in modo indipendente e quasi simultaneamente dagli scienziati di tre delle nazioni coinvolte nella Seconda Guerra Mondiale. Il primo computer *funzionante* fu l'elettromeccanico Heath Robinson,⁹ costruito nel 1943 dal gruppo di Alan Turing con un solo scopo: decifrare i messaggi tedeschi. Sempre nel 1943, lo stesso gruppo sviluppò il Colossus, una potente macchina a uso generale basata su tubi a vuoto.¹⁰ Il primo computer *programmabile* funzionante fu lo Z-3, invenzione di Konrad Zuse nella Germania del 1941. Zuse inventò anche i numeri in virgola mobile e il primo linguaggio di programmazione ad alto livello, Plankalkül. Il primo computer *elettronico*, l'ABC, fu assemblato da John Atanasoff e dal suo studente Clifford Berry tra il 1940 e il 1942 alla Iowa State University. La ricerca di Atanasoff ricevette poco supporto e riconoscimento; il più influente antenato dei computer moderni si rivelò essere l'ENIAC, sviluppato come parte di un progetto militare segreto all'Università della Pennsylvania da una squadra che includeva John Mauchly e J. Presper Eckert.

Da allora in poi, ogni generazione di hardware per computer ha portato un incremento di velocità e potenza e un abbassamento del prezzo – tendenza catturata nella **legge di Moore**. Le prestazioni sono raddoppiate ogni 18 mesi circa fino al 2005, quando problemi di dissipazione dell'energia portarono i produttori a iniziare a moltiplicare il numero dei core delle CPU, anziché aumentare la velocità di clock. Oggi ci si attende che in futuro gli aumenti di funzionalità arriveranno dal parallelismo massivo – curiosa convergenza con le proprietà del cervello. Vediamo anche nuovi progetti hardware basati sull'idea che, per affrontare un mondo incerto, non servano 64 bit di precisione nei numeri: bastano 16 bit (come nel formato bfloat16) o perfino 8 bit, e ciò consentirà un'elaborazione più rapida.

Stiamo appena cominciando a vedere hardware messo a punto per applicazioni di IA come GPU (*graphical processing unit*), TPU (*tensor processing unit*) e WSE (*wafer scale engine*). Dagli anni 1960 a circa il 2012, la potenza di calcolo usata per addestrare i sistemi per le principali applicazioni di apprendimento automatico ha seguito la legge di Moore. A partire dal 2012 le cose sono cambiate: dal 2012 al 2018 vi è stato un aumento di 300.000 volte, che significa un raddoppio ogni 100 giorni circa (Amodei e Hernandez, 2018). Un modello di apprendimento automatico che richiedeva un giorno intero di addestramento nel 2014 richiede soltanto 2 minuti nel 2018 (Ying *et al.*, 2018). Il **computing quantistico** non è ancora praticabile in concreto, ma promette di realizzare eccezionali accelerazioni per alcune importanti sottoclassi di algoritmi di IA.

legge di Moore

Naturalmente sono esistiti dispositivi di calcolo ben prima del computer elettronico. Abbiamo discusso le più antiche macchine automatiche nel Paragrafo 1.2.1. La prima macchina *programmabile* fu un telaio inventato nel 1805 da Joseph Marie Jacquard (1752–1834), che

computing quantistico

⁹ Una macchina complessa il cui nome richiamava quello di un disegnatore di vignette inglese famoso per le sue rappresentazioni di macchine bizzarre e assurdamente complicate il cui scopo era eseguire attività quotidiane, come spalmare il burro su un toast.

¹⁰ Nel dopoguerra Turing voleva usare questi computer per fare ricerca nel campo dell'IA, per esempio per sviluppare uno dei primi programmi di scacchi (Turing *et al.*, 1953). I suoi sforzi furono ostacolati dal governo britannico.

usava schede perforate per memorizzare complesse istruzioni sugli effetti e le decorazioni da incorporare nel tessuto.

A metà del XIX secolo, Charles Babbage (1792–1871) progettò due macchine di calcolo, senza però riuscire a completarle. La macchina alle differenze (*difference engine*) aveva lo scopo di calcolare tabelle matematiche a uso dell'ingegneria e dei progetti scientifici. È stata finalmente costruita e se ne è dimostrato il corretto funzionamento nel 1991 (Swade, 2000). La macchina analitica (*analytical engine*) di Babbage era molto più ambiziosa: comprendeva una memoria indirizzabile, programmi memorizzati basati sulle schede perforate di Jacquard e salti condizionali. Fu la prima macchina capace di calcoli universali. Ada Lovelace, collega di Babbage e figlia del poeta Lord Byron, ne comprese la potenzialità, descrivendola come “una macchina che pensa o... che ragiona”, capace di ragionare su “tutti i temi dell'universo”. Ada Lovelace anticipò anche i cicli di interesse per l'IA, scrivendo: “Bisogna guardarsi dalla possibilità di esagerazioni che potrebbero nascere riguardo i poteri della macchina analitica”. Sfortunatamente le macchine di Babbage e le idee di Lovelace sono state in gran parte dimenticate.

L'IA è in debito anche con la parte software degli studi informatici, che ha messo a disposizione sistemi operativi, linguaggi di programmazione e gli strumenti necessari per scrivere programmi moderni (nonché molti articoli sull'argomento). Ma questa è un'area in cui il debito è stato ripagato: gli studi di IA hanno esplorato per la prima volta idee che si sono poi diffuse nell'informatica generale, tra cui la divisione di tempo, gli interpreti interattivi, i personal computer dotati di finestre e mouse, gli ambienti di sviluppo rapido, la struttura dati chiamata lista concatenata, la gestione automatica della memoria e altri concetti chiave della programmazione simbolica, funzionale, dinamica e orientata agli oggetti (*object-oriented*).

1.2.7 Teoria del controllo e cibernetica

- Come possono degli artefatti funzionare autonomamente?

Ctesibio di Alessandria (circa 250 a.C.) costruì la prima macchina a controllo autonomo: un orologio ad acqua dotato di un regolatore che manteneva costante il flusso d'acqua. Questa invenzione cambiò la definizione di quello che potevano eseguire gli artefatti. In precedenza, solo gli esseri viventi potevano modificare il loro comportamento per reagire a cambiamenti nell'ambiente. Altri esempi di sistemi di controllo che si “auto-regolano” attraverso la retroazione (feedback) includono il regolatore per motori a vapore di James Watt (1736–1819) e il termostato inventato da Cornelis Drebbel (1572–1633), che è anche l'inventore del sottomarino. James Clerk Maxwell (1868) iniziò la teoria matematica dei sistemi di controllo.

Una figura centrale nello sviluppo post bellico della **teoria del controllo** fu Norbert Wiener (1894–1964), un brillante matematico che lavorò tra gli altri con Bertrand Russell, prima di sviluppare un interesse per i sistemi di controllo biologici e meccanici e la loro relazione con la cognizione. Come Craik (che pure usò i sistemi di controllo come modelli psicologici), Wiener e i suoi colleghi Arturo Rosenblueth e Julian Bigelow sfidarono l'ortodossia behaviorista (Rosenblueth *et al.*, 1943). Nella loro concezione, il comportamento volontario scaturiva da un meccanismo di regolazione che cerca di minimizzare l’“errore”, ovvero la differenza tra lo stato corrente del sistema e quello desiderato. Alla fine degli anni 1940 Wiener, insieme a Warren McCulloch, Walter Pitts e John von Neumann, organizzò una serie di importanti convegni dedicati ai nuovi modelli cognitivi matematici e computazionali. Il libro di Wiener *Cybernetics* (**cibernetica**, 1948) divenne un bestseller e rivelò al grande pubblico la possibilità di realizzare macchine intelligenti.

Nel frattempo, in Inghilterra, W. Ross Ashby (Ashby, 1940) faceva da pioniere con concetti simili. Ashby, Alan Turing, Grey Walter e altri formarono il Ratio Club per “coloro che avevano avuto le idee di Wiener prima della pubblicazione del suo libro”. L'opera *Design for a Brain* di Ashby (1948, 1952) sviluppò l'idea secondo cui l'intelligenza poteva essere

teoria del controllo

cibernetica

creata mediante l'uso di dispositivi **omeostatici** contenenti opportuni cicli di feedback per raggiungere un comportamento adattativo stabile.

omeostatico

La moderna teoria del controllo, in particolare la branca nota come controllo ottimo stocastico, ha come scopo la progettazione di sistemi che massimizzano nel tempo una **funzione di costo**. Questo a grandi linee corrisponde al modello standard di IA: la costruzione di sistemi che agiscono in modo ottimo. Ma allora perché l'IA e la teoria del controllo sono due discipline differenti, nonostante l'affinità dei rispettivi fondatori? La risposta sta nelle diverse tecniche matematiche padroneggiate dagli studiosi dei due campi e nei diversi insiemi di problemi affrontati dai due punti di vista. L'analisi matematica e l'algebra delle matrici, gli strumenti della teoria del controllo, si applicano bene a sistemi che si possono descrivere con insiemi finiti di variabili continue, mentre l'IA fu fondata, in parte, proprio per superare queste limitazioni percepite. Strumenti come l'inferenza logica e la computazione permisero agli studiosi di IA di affrontare problemi, come il linguaggio naturale, la visione e la pianificazione simbolica, che si ponevano completamente fuori dal campo d'azione della teoria del controllo.

funzione di costo

1.2.8 Linguistica

- Qual è il collegamento tra linguaggio e pensiero?

Nel 1957 B. F. Skinner pubblicò *Verbal Behavior* (trad. it. *Il comportamento verbale*): si trattava di un resoconto vasto e dettagliato dell'approccio behaviorista all'apprendimento del linguaggio, scritto dal più grande esperto del campo. Curiosamente, una recensione del libro divenne famosa quanto il libro stesso, ed ebbe come conseguenza la quasi totale scomparsa del behaviorismo. L'autore della recensione era il linguista Noam Chomsky, che aveva appena pubblicato un lavoro dedicato alla propria teoria, *Syntactic Structures* (trad. it. *Le strutture della sintassi*). Chomsky evidenziò che la teoria behaviorista non considerava l'aspetto creativo del linguaggio: non spiegava com'è possibile che i bambini potessero comprendere e creare frasi che non hanno mai sentito prima. La teoria di Chomsky, basata su modelli sintattici che risalgono al linguista indiano Panini (circa 350 a.C.) poteva spiegare la creatività, e a differenza delle teorie precedenti era abbastanza formale da poter essere, in via di principio, programmata.

La linguistica moderna e l'IA, quindi, sono "nate" nello stesso momento, e sono cresciute insieme, mescolandosi in un campo di studi ibrido denominato **linguistica computazionale** o **elaborazione del linguaggio naturale**. Il problema della comprensione del linguaggio si rivelò presto molto più complesso di quanto sembrasse nel 1957, perché richiede la comprensione dell'argomento trattato e del contesto, e non solamente della struttura delle frasi. Oggi questo può sembrare ovvio, ma non fu pienamente compreso fino agli anni 1960. Gran parte del lavoro iniziale sulla **rappresentazione della conoscenza** (lo studio di come esprimere la conoscenza in una forma che possa essere usata da un computer per ragionare) fu collegata al linguaggio e si appoggiò alla ricerca dei linguisti, che a loro volta facevano riferimento a decenni di lavoro sull'analisi filosofica del linguaggio.

linguistica computazionale

1.3 La storia dell'intelligenza artificiale

Un modo rapido per riepilogare le tappe fondamentali nella storia dell'IA è quello di elencare i vincitori del Turing Award: Marvin Minsky (1969) e John McCarthy (1971) per la definizione dei fondamenti del campo in base alla rappresentazione e il ragionamento; Ed Feigenbaum e Raj Reddy (1994) per lo sviluppo di sistemi esperti in grado di codificare la conoscenza umana per risolvere problemi del mondo reale; Judea Pearl (2011) per lo sviluppo di tecniche di ragionamento probabilistico in grado di affrontare le situazioni di incertezza in modo formalizzato; e infine Yoshua Bengio, Geoffrey Hinton e Yann LeCun

(2019) per aver fatto del “deep learning” (reti neurali multistrato) una parte fondamentale dell’informatica moderna. Nel seguito di questo paragrafo descriviamo in maggiore dettaglio ogni fase della storia dell’IA.

1.3.1 La gestazione dell’intelligenza artificiale (1943–1956)

Il primo lavoro oggi generalmente considerato appartenente all’IA fu svolto da Warren McCulloch e Walter Pitts (1943). Ispirati dal lavoro di modellazione matematica del docente di riferimento di Pitts, Nicolas Rashevsky (1936, 1938), i due autori si rifecero a tre fonti: la conoscenza delle basi della fisiologia e della funzione dei neuroni nel cervello, un’analisi formale della logica proposizionale di Russell e Whitehead e la teoria della computazione di Turing. I due proposero un modello di neuroni artificiali in cui ogni neurone era caratterizzato dallo stato “acceso” o “spento”, e la cui accensione si verificava in risposta allo stimolo da parte di un numero sufficiente di neuroni adiacenti. Lo stato del neurone veniva così concepito come “di fatto equivalente alla proposizione corrispondente agli stimoli adeguati”. McCulloch e Pitts mostrarono, tra le altre cose, che ogni funzione computabile poteva essere calcolata da una rete di neuroni collegati e che tutti gli operatori logici (AND, OR, NOT e così via) potevano essere implementati con semplici strutture a rete. Suggerirono inoltre che reti neurali adeguatamente definite potessero essere capaci di apprendere. Donald Hebb (1949) formulò una semplice regola di aggiornamento per la modifica dei pesi delle connessioni tra i neuroni: la sua regola, chiamata oggi **apprendimento hebbiano**, rimane un modello importante.

apprendimento
hebbiano

Marvin Minsky (1927–2016) e Dean Edmonds, due studenti di Harvard, costruirono il primo computer basato su reti neurali nel 1950. Lo SNARC, com’era chiamato, utilizzava 3000 tubi a vuoto e un sistema automatico di pilotaggio riciclato da un bombardiere B-24 per simulare una rete di 40 neuroni. Più tardi, a Princeton, Minsky studiò la computazione universale nelle reti neurali. La commissione di dottorato di Minsky non era convinta che questo tipo di lavoro si potesse considerare matematica, ma sembra che von Neumann abbia ribattuto: “Se non lo è ora, lo sarà un giorno”.

Ci furono diversi altri esempi di lavori che possono essere considerati come relativi all’IA, tra cui due programmi per il gioco della dama sviluppati in modo indipendente nel 1952 da Christopher Strachey all’Università di Manchester e da Arthur Samuel in IBM, ma la visione di Alan Turing fu quella con la maggiore influenza. Turing tenne lezioni sull’argomento già nel 1947 presso la London Mathematical Society e articolò un programma di studi persuasivo nel suo articolo del 1950 *Computing Machinery and Intelligence*, in cui introdusse il test di Turing, l’apprendimento automatico, gli algoritmi genetici e l’apprendimento per rinforzo. Affrontò molte delle obiezioni relative alla possibilità di realizzare l’IA e suggerì che sarebbe stato più facile creare IA di livello umano sviluppando algoritmi di apprendimento e poi insegnando alla macchina, anziché programmando manualmente l’intelligenza della macchina. In lezioni successive avvisò che raggiungere questo scopo poteva essere pericoloso per il genere umano.

Nel 1955 John McCarthy del Dartmouth College convinse Minsky, Claude Shannon e Nathaniel Rochester ad aiutarlo a riunire i ricercatori americani interessati alla teoria degli automi, alle reti neurali e allo studio dell’intelligenza. Essi organizzarono un workshop di due mesi a Dartmouth nell’estate del 1956. Parteciparono in 10, tra cui Allen Newell e Herbert Simon di Carnegie Tech,¹¹ Trenchard More di Princeton, Arthur Samuel di IBM, Ray Solomonoff e Oliver Selfridge del MIT. Ecco il testo della proposta:¹²

¹¹ Ora Carnegie Mellon University (CMU).

¹² Questa fu la prima occasione in cui venne utilizzato ufficialmente il termine *intelligenza artificiale*, coniato da McCarthy. Forse un termine come “razionalità computazionale” sarebbe stato più preciso e meno minaccioso, ma “IA” ebbe la meglio. Nel cinquantesimo anniversario della conferenza di Dartmouth, McCarthy affermò di aver evitato i termini “computer” o “computazionale” per rispetto verso Norbert Weiner, che stava promuovendo dispositivi cibernetici analogici anziché computer digitali.

Proponiamo di svolgere uno studio sull'IA per 2 mesi, con 10 persone, durante l'estate del 1956 al Dartmouth College di Hanover, nel New Hampshire. Lo studio procederà sulla base della congettura che ogni aspetto dell'apprendimento, o qualsiasi altra caratteristica dell'intelligenza, possa in linea di principio essere descritto con precisione tale che sia possibile costruire una macchina per simularlo. Si tenterà di scoprire come costruire macchine in grado di utilizzare il linguaggio, formare astrazioni e concetti, risolvere tipi di problemi che oggi sono di esclusiva competenza degli esseri umani, migliorare se stesse. Riteniamo che sia possibile ottenere un significativo progresso in uno o più di questi problemi dedicando un'intera estate al lavoro collettivo di un gruppo di scienziati selezionati.

Nonostante le previsioni ottimistiche, il workshop di Dartmouth non portò a particolari innovazioni. Il lavoro forse più maturo fu quello presentato da Newell e Simon, un sistema per la dimostrazione di teoremi matematici denominato Logic Theorist (LT). Simon dichiarò: “Abbiamo inventato un programma per computer capace di pensare in modo non numerico, e abbiamo quindi risolto l'antichissimo problema mente-corpo”.¹³ Subito dopo il workshop, il programma fu in grado di dimostrare la maggior parte dei teoremi nel secondo capitolo dei *Principia Mathematica* di Russell e Whitehead. Si dice che Russell fosse deliziato quando gli fu detto che Logic Theorist aveva escogitato una dimostrazione di un teorema più breve di quella inclusa nei *Principia*. Gli editor del *Journal of Symbolic Logic* furono meno impressionati e rifiutarono l'articolo che indicava come coautori Newell, Simon e Logic Theorist.

1.3.2 Primi entusiasmi, grandi aspettative (1952–1969)

L'establishment intellettuale degli anni 1950 in linea di massima preferiva credere che “una macchina non potrà mai fare *X*” (nel Capitolo 27 è riportato un lungo elenco di *X* raccolte da Turing). I ricercatori in IA naturalmente risposero dimostrando una *X* dopo l'altra. Si concentrarono in particolare su attività considerate indicative di intelligenza nell'uomo, come giochi, puzzle, problemi matematici e test del quoziante intellettuivo (QI). John McCarthy battezzò questo periodo come l'era del “Guarda mamma, senza mani!”.

Newell e Simon dopo il successo con il Logic Theorist svilupparono il General Problem Solver, o GPS. A differenza del Logic Theorist, questo programma era stato progettato fin dal principio per imitare i procedimenti umani di risoluzione dei problemi. All'interno della classe limitata dei problemi che poteva risolvere, l'ordine in cui il programma considerava i sotto-obiettivi e le possibili azioni era simile a quello adottato dagli esseri umani. GPS quindi fu probabilmente il primo programma ad adottare l'approccio del “pensare umanamente”. Il successo di GPS e dei suoi successori come modelli della cognizione portarono Newell e Simon (1976) a formulare la famosa ipotesi del **sistema fisico di simboli**, che afferma che “un sistema fisico di simboli ha i mezzi necessari e sufficienti per agire in modo generalmente intelligente”. Quello che intendevano è che ogni sistema (umano o artificiale) che dimostra di possedere intelligenza deve funzionare manipolando strutture dati composte di simboli. Vedremo in seguito che quest'ipotesi è stata contestata da più parti.

**sistema fisico
di simboli**

All'IBM, Nathaniel Rochester e i suoi colleghi svilupparono alcuni dei primi programmi di IA. Herbert Gelernter (1959) scrisse il Geometry Theorem Prover, capace di dimostrare teoremi che avrebbero dato qualche grattacapo a molti studenti di matematica. Questo lavoro fu un precursore dei moderni programmi per la dimostrazione di teoremi matematici.

Di tutti i lavori pionieristici svolti in questo periodo, il più influente nel lungo termine fu quello di Arthur Samuel sul gioco della dama. Usando metodi che oggi chiameremmo di apprendimento per rinforzo (cfr. Capitolo 22 del Volume 2), il programma di Samuel imparò a

¹³ Newell e Simon inventarono anche un linguaggio per l'elaborazione di liste, IPL, per scrivere LT. Non avevano compilatore, e dovevano tradurlo a mano in codice macchina. Per evitare di fare errori lavoravano in parallelo, dicendo insieme ad alta voce i numeri binari corrispondenti a ogni istruzione per assicurarsi che coincidessero.

giocare al livello di un buon dilettante. Durante lo sviluppo Samuel dimostrò la falsità dell’idea che i computer possano fare solo ciò che viene loro detto di fare: il software infatti imparò presto a giocare meglio del suo creatore. Il programma venne mostrato alla televisione nel 1956, suscitando forte impressione. Come per Turing, anche per Samuel fu difficile trovare tempo di computazione sui computer allora disponibili: lavorando di notte, poté utilizzare le macchine che si trovavano ancora in fase di test nella fabbrica dell’IBM. Il programma di Samuel fu il precursore di sistemi successivi quali TD-Gammon (Tesauro, 1992), che arrivò tra i migliori giocatori di backgammon del mondo, e AlphaGo (Silver *et al.*, 2016), che stupì il mondo arrivando a sconfiggere il campione del mondo (umano) di Go (cfr. Capitolo 5).

Lisp

Nel 1958 John McCarthy portò due contributi importanti all’IA. Nel memorandum *MIT AI Lab Memo No. 1* definì il linguaggio di alto livello **Lisp**, destinato a diventare il più importante linguaggio di programmazione per l’IA nei trent’anni a seguire. In un articolo intitolato *Programs with Common Sense*, McCarthy presentò una proposta concettuale per sistemi di IA basati su conoscenza e ragionamento. L’articolo descrive l’Advice Taker, un ipotetico programma che avrebbe racchiuso in sé la conoscenza generale del mondo e avrebbe potuto usarla per ricavare piani d’azione. Il concetto era illustrato con un esempio in cui semplici assiomi logici erano sufficienti per generare un piano che consentiva di guidare un’auto fino all’aeroporto. Il programma era anche progettato in modo da accettare nuovi assiomi durante la normale esecuzione, acquisendo così competenze in nuove aree *senza essere riprogrammato*. Così Advice Taker incarnava il principio fondamentale della rappresentazione della conoscenza e del ragionamento: è utile avere una rappresentazione formale ed esplicita del mondo e del suo funzionamento, ed essere in grado di manipolare queste rappresentazioni per mezzo di processi deduttivi. Quell’articolo influenzò il corso dell’IA e mantiene la sua importanza ancora oggi.

Il 1958 è anche l’anno in cui Marvin Minsky passò al MIT. La sua collaborazione iniziale con McCarthy, comunque, ebbe vita breve: quest’ultimo rivolgeva l’attenzione alla rappresentazione e al ragionamento all’interno di una logica formale, mentre a Minsky interessava soprattutto far funzionare i programmi, tanto che a un certo punto arrivò a sviluppare una prospettiva anti-logica. Nel 1963, McCarthy fondò il laboratorio di IA a Stanford. Il suo progetto di usare la logica per costruire una versione definitiva di Advice Taker poteva avvalersi della scoperta del metodo di risoluzione fatta nel 1965 da J. A. Robinson (si tratta di un algoritmo completo per la dimostrazione di teoremi per la logica del primo ordine; cfr. Capitolo 9). Il lavoro a Stanford si concentrava soprattutto su metodi di uso generale per il ragionamento logico. Tra le applicazioni della logica c’erano i sistemi di domanda-risposta e di pianificazione di Cordell Green (Green, 1969b) e il progetto del robot Shakey, presso il nuovo Stanford Research Institute (SRI). Quest’ultimo (che approfondiremo nel Capitolo 26 del Volume 2) fu il primo progetto a dimostrare un’integrazione completa tra il ragionamento logico e l’attività fisica.

micromondo

Al MIT, Minsky supervisionò un gruppo di studenti che scelse di occuparsi di problemi circoscritti la cui soluzione sembrava richiedere un certo grado di intelligenza. Questi domini limitati divennero famosi come **micromondi**. Il programma SAINT (1963) di James Slagle fu capace di risolvere problemi di integrali chiusi tipici dei corsi universitari di primo anno. Il programma ANALOGY (1968) di Tom Evans poteva risolvere problemi basati su analogie geometriche come quelli che appaiono nei test per il quoziente intellettivo. Il programma STUDENT (1967) di Daniel Bobrow risolveva problemi di algebra espressi in forma di racconto, come il seguente:

Se il numero di clienti di Tom è due volte il quadrato del venti per cento del numero degli spot pubblicitari che commissiona, e gli spot commissionati sono 45, quanti clienti avrà Tom?

mondo dei blocchi

Il micromondo più famoso era il **mondo dei blocchi**, che consisteva in una serie di blocchetti solidi disposti su un tavolo (o, più spesso, la sua simulazione), come si può vedere nella Figura

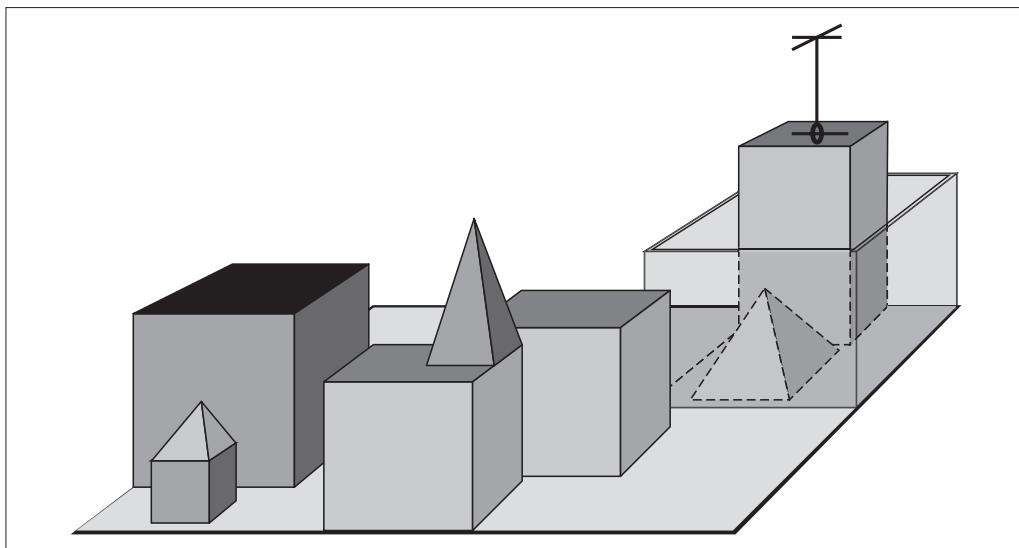


Figura 1.3 Una scena nel mondo dei blocchi. Il programma SHRDLU (Winograd, 1972) ha appena completato l'esecuzione del comando "trova un blocco più alto di quello che stai portando e spostalo all'interno della scatola".

1.3. Un obiettivo tipico, in questo mondo, è sistemare i blocchi in una certa configurazione, usando un braccio meccanico che può prendere un blocco per volta. Il mondo dei blocchi fu l'ambiente sperimentale per il progetto di visione artificiale di David Huffman (1971), il lavoro di David Waltz (1975) dedicato alla visione e alla propagazione dei vincoli, la teoria dell'apprendimento di Patrick Winston (1970), il programma per la comprensione del linguaggio naturale di Terry Winograd (1972) e il pianificatore di Scott Fahlman (1974).

Anche i primi lavori sulle reti neurali di McCulloch e Pitts vennero ripresi e approfonditi: il lavoro di Shmuel Winograd e Jack Cowan (1963) mostrò in che modo si poteva usare un grande numero di elementi per rappresentare collettivamente un singolo concetto, con un incremento di robustezza e parallelismo. I metodi di apprendimento di Hebb furono migliorati da Bernie Widrow (Widrow e Hoff, 1960; Widrow, 1962), che battezzò le proprie reti *adaline*, e da Frank Rosenblatt (1962) con i percettroni. Il teorema di convergenza del percettrone (Block *et al.*, 1962) afferma che l'algoritmo di apprendimento può modificare la forza delle connessioni di un percettrone in modo da corrispondere a qualsiasi possibile input, a patto che tale corrispondenza esista.

1.3.3 Una dose di realtà (1966–1973)

Fin dagli inizi, i ricercatori nel campo dell'IA non hanno avuto paura di fare predizioni sui loro futuri successi. Si cita spesso il seguente brano di Herbert Simon, risalente al 1957:

Il mio scopo non è stupire o sbalordire – ma il modo più semplice in cui posso esprimermi è dicendo che ora nel mondo esistono macchine che possono pensare, imparare e creare. Inoltre, la loro abilità nel fare queste cose aumenterà rapidamente finché, in un futuro vicino, il campo dei problemi che potranno gestire avrà la stessa estensione di quello a cui si è applicata la mente umana.

Il termine "futuro vicino" è piuttosto vago, ma Simon ha anche fatto predizioni più concrete: che entro dieci anni un computer sarebbe stato campione mondiale di scacchi, e che una macchina avrebbe dimostrato un teorema matematico di notevole importanza. Queste predizioni si sono (approssimativamente) avvocate in quarant'anni anziché dieci. L'eccessiva sicurezza di Simon era dovuta alle promettenti prestazioni dei primi sistemi di IA applicati a semplici esempi. In quasi tutti i casi, però, tali sistemi hanno fallito nell'affrontare problemi più complessi.

Questi fallimenti si dovevano a due motivi principali. Il primo è che molti dei primi sistemi di IA si basavano principalmente su una sorta di “introspezione informata” riguardo il modo in cui gli umani svolgono un’attività, anziché su un’attenta analisi dell’attività in questione, delle caratteristiche di una soluzione e di che cosa avrebbe dovuto fare un algoritmo per produrre in modo affidabile tali soluzioni.

Il secondo motivo di fallimento fu l’incapacità di capire l’intrattabilità di molti dei problemi che l’IA stava cercando di risolvere. Molti dei primi programmi di risoluzione dei problemi operavano provando varie combinazioni di passi fino a trovare la soluzione. Questa strategia inizialmente funzionava, perché i micromondi contenevano pochi oggetti e quindi le azioni possibili erano molto poche e le sequenze risolutive corte. Prima dello sviluppo della teoria della complessità computazionale, l’idea diffusa era che la scalabilità verso problemi di maggiori dimensioni fosse solo una questione di hardware più potente e memorie più grandi. L’ottimismo che aveva accompagnato lo sviluppo dei dimostratori automatici di teoremi, per esempio, fu presto smorzato quando i ricercatori non riuscirono a dimostrare teoremi che coinvolgessero più di qualche dozzina di fatti. *Il fatto che un programma possa, in via di principio, trovare una soluzione, non significa affatto che il programma contenga i meccanismi necessari per trovarla nella pratica.*



evoluzione
automatica

L’illusione di una potenza computazionale illimitata non rimase confinata ai programmi dedicati alla risoluzione di problemi. I primi esperimenti di **evoluzione automatica** (ora chiamata **programmazione genetica**) (Friedberg, 1958; Friedberg *et al.*, 1959) erano basati sull’idea, certamente corretta, che apportando una serie appropriata di piccole modifiche a un programma, sarebbe stato possibile generarne un altro con prestazioni particolarmente buone per una qualsiasi particolare attività. L’idea era di provare delle mutazioni casuali applicando poi un processo di selezione per mantenere solo quelle che sembravano utili. Nonostante migliaia di ore di tempo della CPU, fu praticamente impossibile registrare alcun progresso.

La mancanza di una soluzione al problema della “esplosione combinatoria” fu una delle critiche principali all’IA contenute nel rapporto Lighthill (Lighthill, 1973), che fu alla base della decisione, da parte del governo britannico, di tagliare i finanziamenti per l’IA in tutte le università tranne due (la tradizione orale racconta una storia piuttosto differente e molto più colorita, con ambizioni politiche e contrasti personali, la cui narrazione va oltre i nostri scopi).

Un terzo problema sorse a causa di alcuni limiti fondamentali delle strutture di base usate per generare un comportamento intelligente. Per esempio, il libro di Minsky e Papert *Percetrons* (1969) dimostrò che, benché i percetroni (una forma semplice di reti neurali) potevano apprendere tutto ciò che erano in grado di rappresentare, potevano in effetti rappresentare ben poco. In particolare, un percettrone a due input non poteva imparare a distinguere quando i suoi due input erano diversi. Benché questi risultati non si applicassero a reti più complesse a molti livelli, il finanziamento alla ricerca sulle reti neurali presto si ridusse fin quasi a zero. Ironicamente i nuovi algoritmi di retropropagazione per l’apprendimento, che avrebbero causato un grande ritorno alla ricerca sulle reti neurali alla fine degli anni 1980 e poi ancora negli anni 2010, erano in realtà già stati scoperti in altri contesti nei primi anni 1960 (Kelley, 1960; Bryson, 1962).

1.3.4 Sistemi esperti (1969–1986)

metodo debole

La visione generale della risoluzione di problemi, che aveva preso forma durante il primo decennio di studi sull’IA, era quella di un meccanismo di ricerca di uso generale che cercava di mettere insieme dei passi elementari di ragionamento, uno dopo l’altro, fino a trovare soluzioni complete. Questi approcci sono stati chiamati **metodi deboli** perché, sebbene siano generali, non sono in grado di “scalare” verso l’alto e risolvere istanze di problemi molto

grandi o difficili. L'alternativa ai metodi deboli è usare conoscenza più potente, specifica del dominio, che permette di intraprendere passi di ragionamento più ampi e può gestire più facilmente i casi tipici di aree ristrette di esperienza. Si potrebbe dire che, per risolvere un problema difficile, bisogna quasi conoscerne già la soluzione.

Il programma DENDRAL (Buchanan *et al.*, 1969) è una delle prime applicazioni di quest'approccio. Fu sviluppato a Stanford, quando Ed Feigenbaum (un ex-studente di Herbert Simon), Bruce Buchanan (un filosofo trasformato in informatico) e Joshua Lederberg (un genetista vincitore del premio Nobel) unirono i loro sforzi per risolvere il problema della ricostruzione della struttura molecolare partendo dai dati forniti da uno spettrometro di massa. L'input del programma consisteva nella formula elementare di una molecola (per esempio $C_6H_{13}NO_2$) e in uno spettro di massa, che forniva le masse dei vari frammenti di molecola generati in seguito a un bombardamento di elettroni. Per esempio, lo spettro di massa avrebbe potuto contenere un picco per $m = 15$, che corrisponde alla massa di un frammento di metile (CH_3).

La versione iniziale e “ingenua” del programma generava tutte le strutture possibili a partire dalla formula, quindi prediceva quale spettro sarebbe stato osservato per ognuna, confrontando infine le predizioni con lo spettro effettivamente misurato. Come si può facilmente intuire, questo problema diventa intrattabile già per molecole di dimensioni moderate. I ricercatori del progetto DENDRAL consultarono dei chimici analitici e scoprirono che il loro metodo di lavoro consisteva nel cercare dei pattern ben conosciuti di picchi nello spettro, che avrebbero indicato delle strutture di occorrenza comune nella molecola. Per esempio, per riconoscere un sottogruppo chetone ($C=O$), che ha peso 28, si usa la seguente regola:

if M è la massa dell'intera molecola e ci sono due picchi in x_1 e x_2 tali che
 (a) $x_1 + x_2 = M + 28$; (b) $x_1 - 28$ è un picco; (c) $x_2 - 28$ è un picco; e
 (d) almeno uno tra x_1 e x_2 è alto
then c'è un sottogruppo chetone

Sapere che la molecola contiene una particolare struttura riduce enormemente il numero di possibili strutture candidate. Secondo i suoi autori, DENDRAL era potente perché comprendeva in sé tutta la conoscenza della spettroscopia di massa, non solo nei suoi principi fondamentali ma in un'efficiente forma di “ricettario di cucina” (Feigenbaum *et al.*, 1971). DENDRAL rappresentò un passo avanti importante perché fu il primo sistema *a conoscenza intensiva* di successo: la sua capacità era dovuta a un grande numero di regole speciali. Nel 1971, Feigenbaum e altri a Stanford diedero inizio all'Heuristic Programming Project (HPP) per investigare come la nuova metodologia dei **sistemi esperti** potesse essere applicata ad altri campi.

sistema esperto

Un successivo e importante sviluppo fu il sistema MYCIN per diagnosticare le infezioni del sangue. Con circa 450 regole, MYCIN poteva offrire prestazioni pari a quelle di molti esperti, e certamente migliori di quelle dei dottori neolaureati. Rispetto a DENDRAL c'erano due grandi differenze. Prima di tutto, MYCIN non conteneva alcun modello teorico generale da cui poter dedurre le regole, che dovevano essere acquisite con una lunga serie di interviste agli esperti, che a loro volta le avevano apprese dai libri, dai colleghi e con l'esperienza personale. In secondo luogo, le regole dovevano rispecchiare l'incertezza intrinsecamente associata alla conoscenza medica: MYCIN incorporava per questo un metodo di calcolo di incertezza denominato **fattori di certezza** (cfr. Capitolo 13) che all'epoca sembrava corrispondere bene alle modalità con le quali i dottori utilizzavano le informazioni disponibili durante il processo diagnostico.

fattore di certezza

Il primo sistema esperto a ottenere successo commerciale, R1, iniziò l'attività presso la Digital Equipment Corporation (McDermott, 1982). Il programma aiutava a configurare gli ordini di nuovi sistemi informatici; nel 1986 fu stimato che R1 consentì all'impresa di risparmiare 40 milioni di dollari l'anno. Nel 1988 il gruppo di IA di DEC aveva messo in opera 40

sistemi esperti, e altri erano in fase di sviluppo. DuPont ne aveva 100 in uso e 500 in fase di sviluppo. Quasi tutte le più grandi imprese statunitensi avevano un gruppo di IA interno e utilizzavano o stavano studiando i sistemi esperti.

L'importanza della conoscenza del dominio divenne evidente anche nel campo della comprensione del linguaggio naturale. Nonostante il successo del sistema SHRDLU di Winograd, i suoi metodi non gli consentivano di essere esteso ad attività più generali: per problemi quali la risoluzione di ambiguità utilizzava semplici regole basate su un'area ristretta: il mondo dei blocchi.

Molti ricercatori, tra cui Eugene Charniak al MIT e Roger Schank a Yale, suggerirono che la comprensione del linguaggio richiedesse una conoscenza generale del mondo nonché un metodo generale per utilizzarla (Schank andò oltre, arrivando a sostenere che “non esiste una cosa come la sintassi”, affermazione che scatenò le ire di molti linguisti, ma ebbe il prezzo di suscitare un’utile discussione). Schank realizzò insieme ai suoi studenti una serie di programmi (Schank e Abelson, 1977; Wilensky, 1978; Schank e Riesbeck, 1981) dedicati alla comprensione del linguaggio naturale. L'enfasi tuttavia non era posta sul linguaggio in sé, ma più che altro sui problemi connessi alla rappresentazione e al ragionamento applicati alla conoscenza richiesta per la sua comprensione.

frame La crescita e la diffusione di applicazioni a problemi reali portò allo sviluppo di un’ampia varietà di strumenti per la rappresentazione della conoscenza e il ragionamento. Alcuni erano basati sulla logica: per esempio Prolog divenne popolare in Europa e in Giappone, e la famiglia di linguaggi PLANNER negli Stati Uniti. Altri, seguendo le strutture che Minsky aveva denominato **frame** (1975), adottarono un approccio più strutturato, raccogliendo fatti che riguardavano particolari tipi di oggetti ed eventi e organizzando tali tipi in una grande gerarchia analoga alla tassonomia biologica.

Nel 1981 il governo giapponese annunciò il progetto “Quinta Generazione”, un ambizioso piano decennale con l’obiettivo di costruire computer intelligenti usando Prolog. Il budget era di oltre 1,3 miliardi di dollari in moneta di oggi. In risposta, gli Stati Uniti formarono la Microelectronics and Computer Technology Corporation (MCC), un consorzio di ricerca che avrebbe dovuto assicurare la competitività nazionale. In entrambi i casi l’IA faceva parte di uno sforzo a largo raggio, che includeva la progettazione di chip e la ricerca sulle interfacce uomo-macchina. In Inghilterra, il rapporto Alvey ripristinò i finanziamenti tagliati al tempo del rapporto Lighthill. Tuttavia, nessuno di questi progetti arrivò mai a raggiungere i suoi ambiziosi obiettivi in termini di nuove capacità dell’IA o di impatto economico.

Globalmente l’industria dell’IA conobbe un boom, passando da pochi milioni di dollari nel 1980 a miliardi di dollari nel 1988, con centinaia di aziende costruttrici di sistemi esperti, sistemi di visione artificiale, robot, software e hardware specializzati per questi scopi.

Poco dopo sopravvenne il periodo che fu chiamato “inverno dell’IA”, durante il quale molte aziende fallirono per l’impossibilità di mantenere le promesse fatte, spesso alquanto stravaganti. Risultò che era difficile costruire e gestire sistemi esperti per domini complessi, in parte perché i metodi di ragionamento usati dai sistemi non erano in grado di affrontare l’incertezza, e in parte perché i sistemi non erano in grado di imparare dall’esperienza.

1.3.5 Il ritorno delle reti neurali (1986–presente)

A metà degli anni 1980 almeno quattro gruppi diversi reinventarono l’algoritmo di apprendimento basato sulla **retropropagazione** sviluppato nei primi anni 1960. L’algoritmo fu applicato a molti problemi di apprendimento in informatica e psicologia, e la pubblicazione dei risultati nel volume *Parallel Distributed Processing* (Rumelhart e McClelland, 1986) suscitò grande scalpore.

conessionista

Questi modelli chiamati **conessionisti** furono considerati da qualcuno in diretta opposizione sia ai modelli simbolici di Newell e Simon, sia all’approccio logicista di McCarthy e al-

tri. Può sembrare ovvio che in qualche modo gli esseri umani manipolino dei simboli: in effetti il libro dell'antropologo Terrence Deacon *The Symbolic Species* (1997) (trad. it. *La specie simbolica*) sostiene che questa sia la *caratteristica peculiare* degli esseri umani. Contro questa tesi, Geoff Hinton, figura di primo piano nel risorgere delle reti neurali durante gli anni 1980 e 2010, descrisse i simboli come “etere luminifero dell'IA”, riferendosi al mezzo – rivelatosi poi inesistente – attraverso il quale, secondo molti fisici del XIX secolo, si propagavano le onde elettromagnetiche. Certamente, molti concetti a cui assegniamo un nome non presentano, a un esame più attento, le condizioni necessarie e sufficienti, definite logicamente, che i pionieri dell'IA speravano di catturare in forma assiomatica. Forse i modelli connessionali formano concetti interni in un modo più fluido e impreciso, più adatto alla confusione del mondo reale. Hanno anche la capacità di imparare dagli esempi: sono in grado di confrontare il loro valore di output predetto con quello reale, nell'ambito di un problema, e di modificare i loro parametri per ridurre la differenza tra previsione e realtà, potendo così migliorare le prestazioni su problemi futuri.

1.3.6 Ragionamento probabilistico e apprendimento automatico (1987–presente)

La fragilità dei sistemi esperti portò a un nuovo approccio, più scientifico, basato sulla probabilità più che sulla logica booleana, sull'apprendimento automatico più che sulla programmazione manuale, e su risultati sperimentali più che su affermazioni filosofiche.¹⁴ Divenne più comune fondarsi su teorie esistenti anziché proporne di interamente nuove, basare le affermazioni su teoremi rigorosi o su una solida metodologia sperimentale (Cohen, 1995) anziché sull'intuito, e mostrare l'attinenza dei metodi alle applicazioni del mondo reale più che a esempi simili a giochi.

Per dimostrare i progressi compiuti divenne la norma fissare dei problemi di benchmark condivisi, come i data set del repository di UC Irvine per l'apprendimento automatico, l'International Planning Competition per gli algoritmi di pianificazione automatica, il corpus LibriSpeech per il riconoscimento vocale, il data set MNIST per il riconoscimento della scrittura manuale, ImageNet e COCO per il riconoscimento di oggetti da immagini, SQuAD per i sistemi di risposta a domande poste in linguaggio naturale, la competizione WMT per sistemi di traduzione automatica, le International SAT Competition per i risolutori di problemi di soddisfacibilità booleana.

L'IA nacque in parte come movimento di protesta contro le limitazioni imposte da campi di ricerca preesistenti come la teoria del controllo e la statistica, ma ora sta abbracciando quegli stessi campi. Come ha scritto David McAllester (1998):

Nei primi tempi dell'IA sembrava plausibile che nuove forme di calcolo simbolico, come i frame e le reti semantiche, rendessero obsoleta gran parte della teoria classica. Questo portò a una forma di isolazionismo per cui l'IA si trovò in gran parte separata dal resto dell'informatica. L'isolamento è ora superato: si riconosce che l'apprendimento automatico non dovrebbe essere disgiunto dalla teoria dell'informazione, il ragionamento incerto dalla modellazione stocastica, la ricerca dall'ottimizzazione e dal controllo classici, il ragionamento automatico dai metodi formali e dall'analisi statica.

¹⁴ Alcuni hanno descritto questo cambiamento come la vittoria dei **neats** (precisini), quelli che pensano che le teorie dell'IA debbano essere fondate sul rigore matematico, nei confronti degli **scruffies** (scapigliati), che invece preferirebbero mettere alla prova un sacco di idee diverse, scrivere un po' di programmi e poi valutare quello che sembra funzionare meglio. Entrambi gli approcci sono importanti: che ora si tenda verso la precisione significa che la disciplina ha raggiunto un certo livello di stabilità e maturità. L'attuale enfasi sul deep learning potrebbe rappresentare una ripresa degli scapigliati.

modello nascosto di Markov

Questo schema è ben esemplificato dal campo del riconoscimento vocale. Negli anni 1970, in questo settore furono provate una gran varietà di architetture e approcci diversi: molte di queste erano realizzate *ad hoc* e alquanto fragili e funzionavano efficacemente soltanto con alcuni esempi accuratamente selezionati. Negli anni 1980 il campo è stato rapidamente dominato dagli approcci basati sui **modelli nascosti di Markov** (HMM, *hidden Markov models*). Di questi sono importanti due aspetti: prima di tutto si fondano su una rigorosa teoria matematica; questo ha permesso ai ricercatori di avvalersi dei risultati ottenuti in decenni di lavoro in altri campi. In secondo luogo questi modelli sono generati mediante un processo di apprendimento basato su una grande mole di dati reali. Questo assicura prestazioni robuste, e gli HMM hanno dato prova di affidabilità sempre maggiore nei test rigorosi condotti negli ultimi anni. Di conseguenza, il riconoscimento vocale, così come il campo attiguo relativo al riconoscimento della scrittura manuale, hanno compiuto la transizione verso un uso quotidiano da parte dell'industria e degli utenti privati. La scienza non affermava che gli esseri umani utilizzassero gli HMM per riconoscere la voce; invece, gli HMM fornivano una struttura matematica per comprendere e risolvere il problema. Vedremo tuttavia, nel Paragrafo 1.3.8, che il deep learning ha portato scompiglio in questa narrazione.

rete bayesiana

Il 1988 fu un anno importante per il collegamento tra l'IA e altri campi quali statistica, ricerca operativa, teoria delle decisioni e teoria del controllo. Il libro *Probabilistic Reasoning in Intelligent Systems*, di Judea Pearl (1988), portò a una nuova considerazione delle probabilità e della teoria delle decisioni nell'IA. Lo sviluppo delle **reti bayesiane** da parte di Pearl portò un formalismo rigoroso ed efficiente per rappresentare la conoscenza incerta e algoritmi pratici per il ragionamento probabilistico. Questi temi sono trattati nei Capitoli da 12 a 16, insieme a sviluppi più recenti che hanno notevolmente incrementato il potere espresivo dei formalismi probabilistici; il Capitolo 20 del Volume 2 descrive metodi per apprendere reti bayesiane e modelli basati sui dati.

Un altro contributo importante nel 1988 fu quello di Rich Sutton che metteva in connessione l'apprendimento per rinforzo – usato nel programma per il gioco della dama di Arthur Samuel negli anni 1950 – con la teoria dei processi decisionali di Markov (MDP, *Markov decision processes*) sviluppata nel campo della ricerca operativa. Seguirono molti lavori sulla connessione tra la ricerca sulla pianificazione in IA e i processi decisionali di Markov, e il campo dell'apprendimento per rinforzo trovò applicazioni nella robotica e nel controllo dei processi, oltre ad acquisire profonde fondamenta teoriche.

Una conseguenza del ritrovato apprezzamento nell'IA per dati, modellazione statistica, ottimizzazione e apprendimento automatico fu la graduale riunificazione di ambiti quali visione artificiale, robotica, riconoscimento vocale, sistemi multiagente ed elaborazione del linguaggio naturale, che si erano in qualche modo separate dal nucleo centrale dell'IA. Il processo di reintegrazione ha portato notevoli benefici sia in termini di applicazioni – per esempio, in questo periodo è aumentata notevolmente la messa in opera di robot di utilità pratica – sia in termini di una migliore comprensione teorica dei problemi centrali dell'IA.

big data

1.3.7 Big data (2001–presente)

Gli importanti progressi compiuti dal punto di vista della potenza di calcolo e lo sviluppo del World Wide Web hanno facilitato la creazione di insiemi di dati (data set) molto grandi, fenomeno spesso indicato con il termine **big data**. Questi data set contengono migliaia di miliardi di parole di testo, miliardi di immagini, miliardi di ore di audio vocale e video, oltre a enormi quantità di dati genomici, dati per il tracciamento di veicoli, dati sui clic, sui social network e così via.

Ciò ha portato allo sviluppo di algoritmi di apprendimento progettati specificamente per trarre vantaggio da questi grandi insiemi di dati. Spesso la maggioranza degli esempi nell'ambito dei big data è *non etichettata*; per esempio, nell'influente lavoro di Yarowsky (1995)

sulla disambiguazione dei significati delle parole, le occorrenze di una parola inglese come “plant” non sono etichettate nel data set per indicare se si riferiscono alla “pianta” o a un “impianto” (due distinti significati del termine inglese). Disponendo di data set sufficientemente grandi, tuttavia, algoritmi di apprendimento ben studiati sono in grado di raggiungere un’accuratezza di oltre il 96% nell’identificare il significato di una parola in una frase. Inoltre, Banko e Brill (2001) sostengono che il miglioramento di prestazioni ottenuto aumentando la dimensione del data set di due o tre ordini di grandezza supera qualsiasi miglioramento ottenibile mettendo a punto l’algoritmo.

Un fenomeno simile sembra verificarsi nelle attività di visione artificiale quali riempire i buchi nelle fotografie – buchi causati da danni di vario tipo o dalla cancellazione di ex amici. Hays e Efros (2007) hanno sviluppato un ottimo metodo per fare ciò attraverso l’inserimento di pixel tratti da immagini simili; hanno determinato che la tecnica funzionava male con un database di migliaia di immagini, ma raggiungeva una buona qualità disponendo di milioni di immagini. Poco dopo, la disponibilità di decine di milioni di immagini nel database ImageNet (Deng *et al.*, 2009) ha portato a una rivoluzione nel campo della visione artificiale.

La disponibilità di big data e lo spostamento verso l’apprendimento automatico aiutò l’IA a recuperare appetibilità commerciale (Havenstein, 2005; Halevy *et al.*, 2009). I big data rappresentarono un fattore determinante nella vittoria di Watson di IBM sui campioni umani nel gioco a quiz Jeopardy! nel 2011, evento che ebbe grande impatto sulla percezione dell’IA da parte del pubblico.

1.3.8 Deep learning (2011–presente)

Il termine **deep learning** si riferisce all’apprendimento automatico realizzato usando molti livelli di elementi computazionali semplici e regolabili. I primi esperimenti con reti di questo tipo furono condotti già negli anni 1970 ed ebbero un certo successo nella forma di **reti neurali convoluzionali** nell’ambito del riconoscimento di cifre scritte manualmente durante gli anni 1990 (LeCun *et al.*, 1995). Soltanto nel 2011, tuttavia, i metodi di deep learning hanno riscosso un successo reale, prima nel riconoscimento vocale e poi nel riconoscimento visivo di oggetti.

deep learning

Nel 2012 nell’ambito competizione basata su ImageNet, che richiedeva di classificare immagini in una categoria tra migliaia disponibili (armadillo, scaffale, cavatappi, ecc.), un sistema di deep learning creato dal gruppo di Geoffrey Hinton all’Università di Toronto (Krizhevsky *et al.*, 2013) esibì un notevolissimo miglioramento rispetto ai precedenti sistemi basati principalmente su funzionalità realizzate artigianalmente. Da allora, i sistemi di deep learning sono arrivati a superare le prestazioni umane in alcuni compiti di visione (e non in altri). Progressi simili sono stati ottenuti nel riconoscimento vocale, nella traduzione automatica, nelle diagnosi mediche e nei giochi. L’uso di una deep network per rappresentare la funzione di valutazione ha contribuito alle vittorie di ALPHAGO sui migliori giocatori umani di Go (Silver *et al.*, 2016, 2017, 2018).

Questi importanti successi hanno portato a una ripresa dell’interesse nei confronti dell’IA da parte di studenti, imprese, investitori, governi, media e del pubblico in generale. Sembra che ogni settimana vi siano notizie di nuove applicazioni di IA che avvicinano o superano le prestazioni umane, spesso accompagnate da speculazioni circa un futuro luminoso o di nuovo oscuro per l’IA.

Il deep learning richiede hardware potente. Una CPU standard è in grado di eseguire 10^9 o 10^{10} operazioni al secondo, ma un algoritmo di deep learning eseguito su hardware specializzato (per esempio GPU, TPU o FPGA) potrebbe arrivare a elaborare fra le 10^{14} e le 10^{17} operazioni al secondo, per lo più operazioni su matrici e vettori ad alto parallelismo. Naturalmente il deep learning richiede anche la disponibilità di grandi quantità di dati di addestramento e di alcune particolari tecniche algoritmiche (cfr. Capitolo 21 del Volume 2).

1.4 Lo stato dell'arte

L'iniziativa *One Hundred Year Study on AI* della Stanford University (nota anche con la sigla AI100) riunisce gruppi di esperti per fornire report sullo stato dell'arte nell'IA. Il report del 2016 (Stone *et al.*, 2016; Grosz e Stone, 2018) conclude che “È lecito attendersi un notevole incremento dell'uso futuro di applicazioni di IA, tra cui automobili a guida autonoma, sistemi di diagnostica e cure mediche, assistenza per anziani” e che “la società si trova ora a un punto cruciale per determinare come mettere in opera tecnologie di IA in modi che promuovano anziché ostacolare valori democratici come libertà, eguaglianza e trasparenza”. AI100 produce anche un indice **AI Index** presso il sito aiindex.org che facilita il compito di seguire i progressi compiuti. Nel seguito riportiamo alcuni aspetti di rilievo tratti dai report del 2018 e del 2019 (si fa riferimento all'anno 2000 come base di confronto, a meno che non sia indicato altrimenti).

AI index

- Pubblicazioni: gli articoli sull'IA sono aumentati di venti volte dal 2010 al 2019, arrivando a circa 20.000 all'anno. La categoria più popolare è stata quella dell'apprendimento automatico (gli articoli sull'apprendimento automatico in arXiv.org sono raddoppiati ogni anno dal 2009 al 2017), seguita da visione artificiale ed elaborazione del linguaggio naturale.
- Sentiment: circa il 70% delle news sull'IA sono neutrali, ma gli articoli con toni positivi sono aumentati dal 12% nel 2016 al 30% nel 2018. Le criticità più comuni sono quelle etiche: riservatezza dei dati e algoritmi distorti.
- Studenti: le iscrizioni ai corsi in IA sono aumentate di 5 volte negli Stati Uniti e di 16 volte a livello internazionale, rispetto al 2010. L'IA è la più popolare tra le specializzazioni dell'informatica.
- Diversità di genere: i docenti di IA nel mondo sono per circa l'80% maschi e 20% femmine. Numeri simili valgono per gli studenti di dottorato e le assunzioni nel settore.
- Conferenze: la partecipazione a NeurIPS è aumentata dell'800% dal 2012, arrivando a 13.500 partecipanti. Altre conferenze hanno visto crescite annuali di circa il 30%.
- Imprese: le startup nel campo dell'IA negli Stati Uniti sono aumentate di 20 volte, arrivando a oltre 800.
- Internazionalizzazione: la Cina pubblica più articoli per anno degli Stati Uniti e quanti l'intera Europa. Tuttavia, negli indici che tengono conto dell'impatto citazionale, gli autori statunitensi sono avanti del 50% rispetto ai cinesi. Singapore, Brasile, Australia, Canada e India sono i paesi in maggiore crescita in termini di numero di assunzioni nel settore IA.
- Visione artificiale: i tassi di errore nel rilevamento di oggetti (raggiunti nella LSVRC, Large-Scale Visual Recognition Challenge) sono migliorati dal 28% nel 2010 al 2% nel 2017, arrivando a superare le prestazioni umane. L'accuratezza nelle risposte relative all'*open-ended visual question answering* (VQA) è migliorata dal 55% al 68% dal 2015, ma rimane inferiore alle prestazioni umane che si attestano all'83%.
- Velocità: il tempo di addestramento per compiti di riconoscimento di immagini è calato di un fattore 100 solo negli ultimi due anni. La potenza di calcolo usata nelle più importanti applicazioni IA raddoppia ogni 3,4 mesi.
- Linguaggio: l'accuratezza nel rispondere alle domande, misurata con la metrica F1 sul data set SQuAD (Stanford Question Answering Dataset), è aumentata da 60 a 95 dal 2015 al 2019; nella variante SQuAD 2 il progresso è stato ancora più rapido, passando da 62 a 90 in un solo anno. Entrambi i punteggi superano quelli dell'uomo.
- Benchmark umani: nel 2019, i sistemi di IA hanno riportato prestazioni pari o superiori a quelle umane nel gioco degli scacchi, Go, poker, Pac-Man, Jeopardy!, rilevamento di oggetti in ImageNet, riconoscimento vocale in un dominio limitato, traduzione da cinese a inglese in un dominio limitato, Quake III, Dota 2, StarCraft II, vari giochi Atari, rile-

vamento del cancro della pelle, rilevamento del cancro alla prostata, ripiegamento proteico, diagnosi della retinopatia diabetica.

Quando (se mai) i sistemi di IA raggiungeranno le prestazioni umane in una vasta gamma di attività? Ford (2018) ha intervistato in merito numerosi esperti di IA ottenendo un'ampia varietà di indicazioni sull'anno in cui questo avverrà, dal 2029 al 2200, con una media di 2099. In un sondaggio simile (Grace *et al.*, 2017) il 50% degli intervistati ha indicato che questo potrebbe accadere nel 2066, ma il 10% ha indicato già il 2025 e pochi hanno risposto: "Mai". Gli esperti sono stati anche suddivisi tra coloro che ritengono che siano necessarie nuove grandi scoperte e altri secondo i quali basterà mettere a punto gli attuali approcci. Tuttavia, non è il caso di prendere troppo seriamente queste previsioni; come ha illustrato Philip Tetlock (2017), per quanto riguarda la previsione di eventi, gli esperti non sono migliori dei dilettanti.

Come funzioneranno i sistemi di IA del futuro? Non siamo ancora in grado di dirlo. Come si è visto, il campo dell'IA ha visto perseguire obiettivi diversi con approcci diversi: prima lo studio dell'idea stessa che fosse possibile dotare le macchine di intelligenza, poi che si potesse codificare la conoscenza degli esperti nella logica formale, poi che lo strumento principale fossero i modelli probabilistici del mondo, e più recentemente che l'apprendimento automatico consentirà di ottenere modelli non basati su alcuna teoria conosciuta. Sarà il futuro a rivelare il prossimo modello.

Che cosa è in grado di fare l'IA oggi? Forse non tanto quanto alcuni degli articoli di giornale più ottimistici potrebbero indurre a credere, ma comunque molto. Nel seguito riportiamo alcuni esempi.

Veicoli robotizzati: la storia dei veicoli robotizzati risale alle automobili radiocontrollate degli anni 1920, ma le prime dimostrazioni di veicoli a guida autonoma nelle strade senza ausili speciali sono state effettuate negli anni 1980 (Kanade *et al.*, 1986; Dickmanns e Zapp, 1987). Dopo dimostrazioni di successo di guida su strade isolate nella DARPA Grand Challenge del 2005 (Thrun, 2006) e su strade trafficate nell'Urban Challenge del 2007, la corsa a sviluppare auto a guida autonoma cominciò a farsi seria. Nel 2018 i veicoli prova di Waymo superarono la soglia dei 10 milioni di miglia guidate su strade pubbliche senza incidenti gravi, con il guidatore umano che interveniva a prendere il controllo del mezzo soltanto una volta ogni 6.000 miglia. Poco dopo, l'impresa ha iniziato a offrire un servizio di taxi robotizzato commerciale.

Nell'aria, i droni autonomi ad ala fissa sono utilizzati per consegnare sacche di sangue in Ruanda già dal 2016. I quadricotteri eseguono manovre acrobatiche notevoli, esplorano edifici per costruirne mappe 3D e si riuniscono in formazioni autonome.

Locomozione su arti: BigDog, un robot quadrupede di Raibert *et al.* (2008), ha stravolto la nostra nozione di movimento dei robot: non più l'andatura lenta, rigida sulle gambe dei robot da film di Hollywood, ma qualcosa di più simile al modo in cui si muovono gli animali, con la capacità di riprendersi in caso di scontri con ostacoli o scivolate su terreni ghiacciati. Atlas, un robot umanoide, non solo cammina su terreni irregolari, ma salta su gradini e fa i salti mortali all'indietro (Ackerman e Guizzo, 2016).

Pianificazione e scheduling autonomo: a centinaia di milioni di chilometri dal pianeta Terra, il programma Remote Agent della NASA è stato il primo programma di pianificazione autonoma a bordo in grado di controllare lo scheduling delle operazioni per un veicolo spaziale (Jonsson *et al.*, 2000). Remote Agent generava piani in base a obiettivi di alto livello indicati da terra e ne monitorava l'esecuzione rilevando, diagnosticando e risolvendo i problemi che si verificavano. Oggi, il sistema di pianificazione EUROPA (Barreiro *et al.*, 2012) è usato per le attività quotidiane dei rover Mars della NASA e il sistema SEXTANT (Winternitz, 2017) consente la navigazione autonoma nello spazio profondo, oltre il raggio d'azione del sistema GPS globale.

Durante la crisi del Golfo del 1991, le forze armate statunitensi misero in opera uno strumento di analisi e ripianificazione dinamica, DART (Cross e Walker, 1994), per svolgere automaticamente compiti di pianificazione logistica e scheduling per i trasporti. Si trattava di gestire fino a 50.000 veicoli, cargo e persone alla volta, dovendo tenere conto di punti di partenza, destinazioni, percorsi, capacità di trasporto a terra, nei porti e in aria, e risoluzione di conflitti tra tutti i parametri. La DARPA (Defense Advanced Research Project Agency) ha affermato che questa applicazione da sola ha più che ripagato 30 anni di investimenti in IA effettuati dall'agenzia.

Ogni giorno società di servizi a chiamata come Uber e servizi di mappe come Google Maps forniscono indicazioni di guida a centinaia di milioni di utenti, tracciando rapidamente un percorso ottimale che tiene conto delle condizioni del traffico attuali e future.

Traduzione automatica: i sistemi di traduzione automatica online oggi consentono di leggere documenti scritti in oltre 100 lingue, incluse quelle native di oltre il 99% della popolazione umana, e forniscono centinaia di migliaia di miliardi di parole ogni giorno a centinaia di milioni di utenti. Benché non siano perfetti, sono generalmente adeguati per una comprensione di base. Per linguaggi più vicini tra loro e con una maggiore quantità di dati disponibili per l'addestramento (come francese e inglese) le traduzioni in domini ristretti sono vicine al livello di un traduttore umano (Wu *et al.*, 2016b).

Riconoscimento vocale: nel 2017, Microsoft ha dimostrato che il suo Conversational Speech Recognition System aveva raggiunto un tasso di errore del 5,1%, raggiungendo la prestazione dell'uomo nell'attività di centralino, che comporta anche la trascrizione di conversazioni telefoniche (Xiong *et al.*, 2017). Circa un terzo delle interazioni con i computer in tutto il mondo oggi è svolto a voce anziché con la tastiera; Skype fornisce servizi di interpretariato automatico in tempo reale in dieci linguaggi. Alexa, Siri, Cortana e Google offrono assistenti in grado di rispondere a domande e svolgere compiti per conto dell'utente; per esempio, il servizio Google Duplex usa il riconoscimento e la sintesi vocale per effettuare prenotazioni al ristorante, conducendo una conversazione fluente per conto dell'utente.

Raccomandazioni: società quali Amazon, Facebook, Netflix, Spotify, YouTube, Walmart e altre usano l'apprendimento automatico per fornire raccomandazioni agli utenti sulla base delle loro esperienze passate e di altri utenti dai gusti simili. Il campo dei sistemi di raccomandazione ha una storia lunga (Resnick e Varian, 1997) ma sta cambiando rapidamente a causa dei nuovi metodi di deep learning che analizzano i contenuti (testo, musica, video) oltre alla cronologia e ai metadati (van den Oord *et al.*, 2014; Zhang *et al.*, 2017). Anche i filtri per lo spam possono essere considerati una forma di sistema di raccomandazione (al contrario); le attuali tecniche di IA sono in grado di filtrare oltre il 99,9% dello spam e i servizi di email possono anche raccomandare potenziali destinatari e possibili testi di risposta.

Giochi: quando Deep Blue sconfisse il campione del mondo di scacchi Garry Kasparov nel 1997, i difensori della supremazia umana riposero le loro speranze nel gioco del Go. Piet Hut, astrofisico e appassionato di Go, predisse che sarebbero servito “un centinaio di anni prima che un computer riuscisse a sconfiggere i giocatori umani a Go – forse anche di più”. Ma soltanto 20 anni dopo, ALPHAGO superò tutti i giocatori umani (Silver *et al.*, 2017). Ke Jie, il campione del mondo, disse: “L’anno scorso era ancora simile all’uomo nel giocare, ma quest’anno è diventato una specie di dio del Go”. ALPHAGO ha tratto vantaggio dallo studio di centinaia di migliaia di partite giocate in passato da giocatori umani di Go, e dalle conoscenze di esperti giocatori di Go che hanno collaborato con il team di sviluppo.

ALPHAZERO, un programma successivo, non usava alcun input umano (a parte le regole del gioco) ed era in grado di apprendere giocando contro se stesso fino a sconfiggere tutti gli avversari – umani e macchine – a Go, scacchi e shogi (Silver *et al.*, 2018). Nel frattempo, campioni umani sono stati sconfitti da sistemi di IA in vari giochi quali Jeopardy! (Ferrucci *et al.*, 2010), poker (Bowling *et al.*, 2015; Moravčík *et al.*, 2017; Brown e Sandholm, 2019) e

nei videogiochi Dota 2 (Fernandez e Mahlmann, 2018), StarCraft II (Vinyals *et al.*, 2019) e Quake III (Jaderberg *et al.*, 2019).

Interpretazione delle immagini: non contenti di superare la precisione dell'uomo nell'attività di riconoscimento di oggetti in ImageNet, gli studiosi di visione artificiale hanno affrontato un problema ancora più difficile: l'*image captioning*, cioè la descrizione di immagini mediante didascalie. Tra gli esempi: “Una persona che percorre in motocicletta una strada isolata”, “Due pizze appoggiate sul piano della cucina” e “Un gruppo di giovani che giocano a frisbee” (Vinyals *et al.*, 2017b). I sistemi attuali, tuttavia, non sono affatto perfetti: un'immagine descritta come “frigorifero con molti alimenti e bevande” si rivela essere un segno di divieto di parcheggiare parzialmente coperto da vari adesivi.

Medicina: gli algoritmi di IA oggi raggiungono o superano medici esperti nella diagnosi di molte malattie, soprattutto quando la diagnosi si basa su immagini. Questo vale per esempio per il morbo di Alzheimer (Ding *et al.*, 2018), il cancro metastatico (Liu *et al.*, 2017; Esteva *et al.*, 2017), le malattie oftalmiche (Gulshan *et al.*, 2016) e quelle della pelle (Liu *et al.*, 2019c). Una revisione sistematica basata su meta-analisi (Liu *et al.*, 2019a) ha determinato che le prestazioni dei programmi di IA, in media, sono risultate equivalenti a quelle dei professionisti sanitari. In campo medico, al momento si pone l'enfasi sull'impiego dell'IA per facilitare l'interazione tra uomo e macchina. Per esempio, il sistema LYNA raggiunge il 99,6% di accuratezza complessiva nella diagnosi di cancro al seno metastatico – meglio di un esperto umano senza aiuti – ma la combinazione con l'uomo fornisce risultati ancora migliori (Liu *et al.*, 2018; Steiner *et al.*, 2018).

L'ampia adozione di queste tecniche oggi è limitata non solo dall'accuratezza diagnostica, ma dalla necessità di dimostrare miglioramenti nei risultati clinici e di garantire trasparenza, assenza di distorsioni e pregiudizi e riservatezza dei dati (Topol, 2019). Nel 2017, soltanto due applicazioni mediche dell'IA sono state approvate dalla FDA, ma il numero è cresciuto a 12 nel 2018 ed è in continuo aumento.

Climatologia: un team di scienziati ha vinto il Gordon Bell Prize del 2018 per un modello di deep learning in grado di rivelare informazioni dettagliate su eventi climatici estremi che in precedenza andavano persi nei dati del clima. Hanno usato un supercomputer con hardware GPU specializzato per arrivare oltre il livello exaop (10^{18} operazioni al secondo); è stato il primo programma di apprendimento automatico a fare questo (Kurth *et al.*, 2018). Rolnick *et al.* (2019) hanno presentato un documento di 60 pagine per catalogare i modi in cui l'apprendimento automatico può essere usato per affrontare il cambiamento climatico.

Quelli riportati qui sopra sono solo alcuni esempi di sistemi di IA esistenti oggi. Non si tratta di fantascienza o magia, ma di scienza, ingegneria e matematica, e questo libro fornisce un'introduzione in merito.

1.5 Rischi e opportunità dell'intelligenza artificiale

Francis Bacon, filosofo accreditato come il creatore del metodo scientifico, osservò nel *De Sapientia Veterum* (1609, trad. it. *Sapienza degli antichi*) che “Le arti meccaniche sono ambigue, possono servire per ferire come per guarire”. Poiché l'IA ha un ruolo sempre più importante in campo economico, sociale, scientifico, medico, finanziario e militare, è importante considerarne i pericoli e i rimedi, o per dirla in parole più attuali, i rischi e le opportunità. I temi accennati qui sono trattati in maggiore dettaglio nei Capitoli 27 e 28.

Iniziamo con le opportunità: in parole semplici, la nostra civiltà è il prodotto dell'intelligenza umana. Se abbiamo la possibilità di accedere a un'intelligenza automatica notevolmente superiore, il limite delle nostre ambizioni si innalza di conseguenza. La potenzialità dell'IA e della robotica di liberare l'umanità dai compiti più umili e ripetitivi e di

aumentare decisamente la produzione di beni e servizi potrebbe far presagire un'era di pace e abbondanza. La capacità di accelerare la ricerca scientifica potrebbe consentire di trovare cure per malattie e soluzioni per i problemi del cambiamento climatico e della scarsità di risorse. Come ha suggerito Demis Hassabis, CEO di Google DeepMind: “Prima risolviamo l'IA, poi usiamola per risolvere tutto il resto”.

Tuttavia, molto prima di avere un'opportunità di “risolvere l'IA”, dovremo affrontare i rischi presentati da usi errati dell'IA, voluti o meno. Alcuni di questi sono già evidenti, altri sembrano solo probabili sulla base delle tendenze attuali.

- *Armi autonome letali*: secondo la definizione delle Nazioni Unite si tratta di armi in grado di localizzare, selezionare ed eliminare bersagli umani senza l'intervento umano. Una delle principali preoccupazioni relative a queste armi è la loro *scalabilità*: l'assenza del requisito della supervisione umana significa che un piccolo gruppo potrebbe impiegare un elevatissimo numero di armi contro bersagli umani definiti da qualsiasi criterio di riconoscimento praticabile. Le tecnologie necessarie per le armi autonome sono simili a quelle necessarie per i veicoli a guida autonoma. Discussioni informali tra esperti sui potenziali rischi delle armi autonome letali sono iniziate alle Nazioni Unite nel 2014, e nel 2017 si è passati alla fase formale di pre-trattato con un “Gruppo di Esperti Governativi”.
- *Sorveglianza e persuasione*: per quanto sia costoso, noioso e a volta discutibile dal punto di vista legale che personale di sicurezza controlli linee telefoniche, videocamere, email e altri canali informativi, l'IA (riconoscimento vocale, visione artificiale, comprensione del linguaggio naturale) può essere usata in modo scalabile per svolgere attività di sorveglianza di massa su individui e rilevare attività di interesse. Ritagliando flussi di informazioni personalizzati per gli individui attraverso i social media e sfruttando tecniche di apprendimento automatico, è possibile anche modificare e controllare in qualche misura il comportamento politico – un rischio che ha cominciato a manifestarsi nelle elezioni a partire dal 2016.
- *Distorsione delle decisioni*: un impiego malevolo, volutamente o meno, degli algoritmi di apprendimento automatico per attività quali la valutazione di domande di semilibertà o di finanziamento, può dare origine a decisioni discriminatorie per etnia, genere o altre categorie protette. Spesso i dati stessi riflettono distorsioni diffuse nella società.
- *Impatto sull'occupazione*: da secoli si riflette sul rischio che le macchine eliminino i posti di lavoro. La storia non è mai semplice: le macchine svolgono alcune delle attività che altrimenti potrebbero essere svolte dagli umani, ma allo stesso tempo rendono gli umani più produttivi e quindi ne favoriscono l'impiego nel lavoro, inoltre rendono le imprese più redditizie e quindi in grado di pagare stipendi più elevati. Possono rendere economicamente praticabili attività che senza di esse non lo sarebbero. L'uso delle macchine solitamente genera maggiore ricchezza, ma tende ad avere l'effetto di spostare la ricchezza dal lavoro al capitale, esacerbando le diseguaglianze. In passato alcuni progressi tecnologici, come l'invenzione del telaio meccanico, hanno portato a gravi problemi di disoccupazione, ma alla fine le persone hanno trovato nuovi tipi di lavori da svolgere. D'altra parte, è possibile che l'IA sarà in grado di svolgere anche questi tipi di lavori. Questo tema sta rapidamente diventando centrale per economisti e governi di tutto il mondo.
- *Sistemi critici per la sicurezza*: grazie ai progressi compiuti, le tecniche di IA sono sempre più utilizzate in applicazioni ad alto impatto e in cui la sicurezza è un fattore critico, come la guida di automobili e la gestione delle forniture idriche delle città. Si sono già verificati incidenti mortali a testimoniare la difficoltà di svolgere verifiche formali e analisi statistiche del rischio per sistemi sviluppati usando tecniche di apprendimento automatico. Nel campo dell'IA si dovranno sviluppare tecniche e standard etici almeno paragonabili a quelli diffusi in altre discipline tecniche e sanitarie dove è in gioco la vita delle persone.

- **Cybersicurezza:** le tecniche di IA sono utili per la difesa contro cyberattacchi, per esempio rilevando schemi di comportamento insoliti, ma possono anche contribuire a generare malware potente, capace di sopravvivere e di proliferare. Per esempio, i metodi di apprendimento per rinforzo sono stati usati per creare strumenti molto efficaci per portare attacchi automatizzati ed estorsioni personalizzate via email e phishing.

Torneremo a esaminare questi argomenti in maggiore dettaglio nel Paragrafo 27.3. I sistemi di IA, diventando sempre più potenti, tenderanno ad assumere sempre di più ruoli che in passato erano svolti esclusivamente da esseri umani. E dato che gli umani hanno usato tali ruoli anche per perpetrare crimini, è lecito attendersi che possano usare i sistemi di IA in quei ruoli per perpetrare crimini ancora maggiori. Tutti gli esempi presentati in precedenza evidenziano l'importanza della governance e anche di una regolamentazione. Attualmente, la comunità di ricerca e le principali imprese coinvolte negli studi sull'IA hanno sviluppato principi di autoregolamentazione su base volontaria per le attività di IA (cfr. Paragrafo 27.3). Governi e organismi internazionali stanno formando gruppi di lavoro per elaborare regole adatte a ciascun caso specifico, per prepararsi ai futuri impatti economici e sociali, e per trarre vantaggio dalle capacità dell'IA per risolvere importanti problematiche sociali.

Che cosa avverrà nel lungo periodo? Raggiungeremo l'obiettivo finale, ovvero la creazione di sistemi dotati di intelligenza comparabile o perfino superiore a quella umana? E in caso positivo, che cosa verrà dopo?

In buona parte della storia dell'IA, queste domande sono state lasciate sullo sfondo, oscure dall'ansia quotidiana di creare sistemi di IA in grado di fare qualsiasi cosa in modo anche lontanamente intelligente. Come avviene per qualsiasi disciplina di ampio respiro, la grande maggioranza di coloro che fanno ricerca in IA si è specializzata in campi specifici quali i giochi, la rappresentazione della conoscenza, la visione artificiale o la comprensione del linguaggio naturale, spesso basandosi sull'assunto che i progressi compiuti in questi campi avrebbero contribuito a raggiungere gli obiettivi più generali dell'IA. Nils Nilsson (1995), uno dei leader originari del progetto Shakey presso SRI, ricordò agli specialisti che esistevano gli obiettivi più generali e avisò che i campi specifici rischiavano di diventare fini a se stessi. In seguito, alcuni influenti fondatori dell'IA, tra cui John McCarthy (2007), Marvin Minsky (2007) e Patrick Winston (Beal e Winston, 2009), concordarono con gli avvertimenti di Nilsson e suggerirono che, invece di focalizzarsi sulle prestazioni misurabili in applicazioni specifiche, l'IA dovesse tornare alle sue origini per puntare, nelle parole di Herb Simon, a realizzare “macchine che pensano, imparano e creano”. Diedero a questo obiettivo il nome **IA a livello umano**, o **HLAI** (*human-level AI*), a indicare che una macchina dovrebbe essere in grado di imparare a fare qualsiasi cosa che possa fare un essere umano. Il primo congresso di questa corrente di pensiero si tenne nel 2004 (Minsky *et al.*, 2004). Un'altra iniziativa con obiettivi simili, il movimento per l'**intelligenza artificiale generale** (**AGI**, *artificial general intelligence*) (Goertzel e Pennachin, 2007), tenne la sua prima conferenza e fondò il *Journal of Artificial General Intelligence* nel 2008.

Più o meno nello stesso periodo nacquero le preoccupazioni circa il fatto che creare una **superintelligenza artificiale** (**ASI**, *artificial superintelligence*) – in grado di superare decisamente le capacità umane – potesse essere una cattiva idea (Yudkowsky, 2008; Omohundro, 2008). Turing stesso (1996) aveva sostenuto ciò in una lezione tenuta a Manchester nel 1951, riprendendo idee precedenti di Samuel Butler (1863):¹⁵

IA a livello umano

**intelligenza
artificiale generale**

**superintelligenza
artificiale**

¹⁵ Ancora prima, nel 1847, Richard Thornton, editor di *Primitive Expounder*, si era scagliato contro i calcolatori meccanici: “La mente ... va oltre se stessa e mette a rischio la necessità della propria esistenza inventando macchine in grado di pensare da sole ... Ma chi può sapere se tali macchine, quando avranno raggiunto un livello più elevato di perfezione, potranno pensare a un piano per correggere tutti i loro difetti e poi elaboreranno idee che andranno oltre la comprensione delle menti mortali!”

Appare probabile che, una volta avviato, il metodo di pensiero automatico non impiegherà troppo tempo per superare i nostri flebili poteri. Dobbiamo aspettarci che prima o poi le macchine assumeranno il controllo, nel modo citato in *Erewhon* di Samuel Butler.

Queste preoccupazioni si sono fatte ancora più serie con i recenti progressi compiuti nel deep learning, la pubblicazioni di libri quali *Superintelligence* di Nick Bostrom (2014, trad. it. *Superintelligenza*) e pronunciamenti pubblici di Stephen Hawking, Bill Gates, Martin Rees ed Elon Musk.

problema del gorilla

È del tutto naturale provare un senso di disagio all'idea di creare macchine superintelligenti. Potremmo chiamarlo **problema del gorilla**: circa sette milioni di anni fa vi fu l'evoluzione di un primate oggi estinto, che portò in un ramo ai gorilla e in un altro agli esseri umani. Oggi i gorilla non sono felici dell'evoluzione umana: in pratica non hanno alcun controllo sul loro futuro. Se questo sarà il risultato della creazione dell'IA superumana, cioè che gli umani cederanno il controllo sul proprio futuro, forse dovremmo smettere di lavorare all'IA e, come corollario, rinunciare ai benefici che potrebbe portare. Questa è l'essenza dell'avvertimento di Turing: non è certo che saremo in grado di controllare macchine che saranno più intelligenti di noi.

Se l'IA superumana fosse una scatola nera arrivata dallo spazio, sarebbe certamente saggio esercitare cautela nell'aprirla. Ma non lo è: siamo *noi* a progettare i sistemi di IA, perciò se essi finiranno per “prendere il controllo”, come suggerisce Turing, questo sarà il risultato di un errore di progettazione. Per evitare ciò, dobbiamo comprendere la fonte di potenziali errori. Norbert Wiener (1960), spinto a considerare il futuro di lungo periodo dell'IA dopo aver visto il programma per gli scacchi di Arthur Samuel imparare a sconfiggere il suo creatore, ebbe a dire questo:

Se, per raggiungere i nostri scopi, utilizziamo un agente meccanico di cui non possiamo realmente controllare il comportamento... faremo bene ad accertarci che lo scopo incorporato nella macchina sia proprio quello che desideriamo.

problema di Re Mida

In molte culture esistono miti di esseri umani che chiedono qualcosa a dei, geni, maghi o diavoli. In tutti i casi essi ottengono ciò che letteralmente chiedono, per poi pentirsene. Il terzo desiderio, se c'è, è quello di annullare i primi due. Lo chiameremo il **problema di Re Mida**: Mida, sovrano leggendario della mitologia greca, chiese che ogni cosa che toccasse fosse trasformata in oro, ma poi se ne pentì quando gli capitò di toccare cibo, bevande e familiari.¹⁶

Abbiamo incontrato questo tema nel Paragrafo 1.1.5, quando abbiamo evidenziato la necessità di una significativa modifica del modello standard che prescrive di inserire obiettivi fissati nella macchina. La soluzione alla questione posta da Wiener non è quella di avere uno “scopo ben definito da inserire nella macchina”. Al contrario, vogliamo macchine che cerchino di raggiungere obiettivi umani sapendo però di non conoscere in modo certo quali siano con esattezza tali obiettivi.

gioco di assistenza

Dobbiamo purtroppo constatare che quasi tutte le ricerche in IA svolte finora sono state portate avanti nell'ambito del modello standard, il che significa che quasi tutti gli argomenti tecnici presentati in questo volume riflettono tale impostazione. Esistono tuttavia alcuni primi risultati ottenuti nel nuovo quadro concettuale. Nel Capitolo 16 mostreremo che una macchina ha un incentivo positivo a consentire a se stessa di farsi spegnere se e solo se è incerta sull'obiettivo dell'uomo. Nel Capitolo 18 formuleremo e studieremo i **giochi di assistenza**, che descrivono matematicamente la situazione in cui un essere umano ha un obiettivo e una macchina cerca di raggiungerlo, ma è inizialmente incerta su quale sia tale obiettivo. Nel Capitolo 22 del Volume 2 spiegheremo i metodi di **apprendimento per rin-**

¹⁶ Mida avrebbe fatto meglio a seguire i principi di base di precauzione e a inserire un pulsante “annulla” e un pulsante “pausa” nei suoi desideri.

apprendimento per rinforzo inverso

forzo inverso, che consentono alle macchine di imparare le preferenze umane dalle osservazioni delle scelte effettuate dagli esseri umani. Nel Capitolo 27 esamineremo due tra le principali difficoltà da superare: il fatto che le nostre scelte dipendono dalle nostre preferenze per il tramite di un'architettura cognitiva molto complessa che è difficile da analizzare per riprodurne i meccanismi, e il fatto che gli esseri umani potrebbero non avere preferenze coerenti – a livello di individuo o di gruppo – perciò potrebbe non essere chiaro che cosa i sistemi di IA *dovrebbero* fare per noi.

1.6 Riepilogo

Questo capitolo ha fornito una definizione dell'IA e ha descritto il background culturale in cui si è sviluppata. Di seguito sono riportati alcuni dei punti più importanti.

- Persone differenti si avvicinano all'IA con obiettivi diversi. Due domande importanti sono: vi interessa più il pensiero o il comportamento? Volete usare come modello gli esseri umani o cercare di raggiungere i risultati ottimali?
- Secondo quello che abbiamo chiamato modello standard, l'IA riguarda principalmente l'**azione razionale**. Un **agente intelligente** ideale intraprende in ogni situazione la migliore azione possibile. Studieremo il problema di costruire agenti che sono “intelligenti” secondo questa particolare accezione.
- Questo concetto semplice richiede due raffinamenti: in primo luogo, la capacità di ogni agente, umano o altro, di scegliere azioni razionali è limitata dall'intrattabilità computazionale di fare ciò; in secondo luogo, il concetto di una macchina che persegua un obiettivo definito deve essere sostituito con quello di una macchina che persegua obiettivi vantaggiosi per gli esseri umani, ma incerta su quali siano tali obiettivi.
- I filosofi, a partire dal 400 a.C., hanno reso concepibile lo sviluppo dell'IA suggerendo che la mente sia per certi aspetti simile a una macchina, che funzioni sulla base di conoscenze rappresentate in qualche forma di linguaggio interno e che si possa usare il pensiero per scegliere quale azione compiere.
- I matematici hanno fornito gli strumenti per manipolare enunciati sia in condizioni di certezza logica che di incertezza probabilistica. Inoltre, hanno sviluppato la teoria necessaria a comprendere la computazione e analizzare gli algoritmi.
- Gli economisti hanno formalizzato il problema di prendere decisioni che massimizzino il risultato atteso del decisore.
- Gli studiosi di neuroscienze hanno scoperto alcuni fatti sul funzionamento del cervello e sui punti di somiglianza e di differenza rispetto ai computer.
- Gli psicologi hanno adottato l'idea che gli esseri umani e gli animali possano essere considerati macchine che elaborano informazioni. I linguisti hanno mostrato che l'uso del linguaggio si adatta a questo modello.
- Gli ingegneri informatici specialisti nell'hardware hanno creato macchine sempre più potenti che rendono possibili le applicazioni dell'IA, e gli ingegneri informatici specialisti nel software hanno reso tali applicazioni più facili da usare.
- La teoria del controllo si occupa della progettazione di dispositivi che agiscono in modo ottimo sulla base della retroazione fornita dall'ambiente. All'inizio gli strumenti matematici della teoria del controllo differivano molto da quelli usati nell'IA, ma ora i due campi si stanno avvicinando.
- La storia dell'IA è stata segnata da cicli in cui al successo ha fatto seguito un eccessivo ottimismo, seguito da una caduta d'entusiasmo e tagli di fondi. Ci sono anche stati cicli in cui l'introduzione di nuovi approcci creativi si è alternata con il raffinamento e la sistematizzazione dei migliori tra essi.

- L'IA è maturata notevolmente rispetto ai primi decenni della sua storia, sia a livello teorico che metodologico. Con l'aumentare della complessità dei problemi che l'IA doveva affrontare, si è passati dalla logica booleana al ragionamento probabilistico, e dalla conoscenza artigianale all'apprendimento automatico basato sui dati. Tutto ciò ha portato a notevoli miglioramenti delle capacità dei sistemi reali e a una maggiore integrazione con altre discipline.
- Con l'aumentare delle applicazioni di sistemi di IA nel mondo reale, è divenuto necessario considerare un'ampia varietà di rischi e conseguenze etiche.
- Nel lungo periodo dobbiamo affrontare il difficile problema di controllare sistemi di IA superintelligenti che potrebbero evolvere in modi imprevedibili. Per risolvere questo problema appare necessario un mutamento nella nostra concezione dell'IA.

Note storiche e bibliografiche

Una storia dell'IA è fornita da Nils Nilsson (2009), uno dei pionieri della disciplina. Pedro Domingos (2015) e Melanie Mitchell (2019) forniscono panoramiche dell'apprendimento automatico rivolte a un pubblico di non esperti del campo, e Kai-Fu Lee (2018) descrive la corsa per ottenere la leadership in IA a livello internazionale. Martin Ford (2018) intervista 23 studiosi leader nell'IA.

Le principali associazioni professionali per l'IA sono l'Association for the Advancement of Artificial Intelligence (AAAI), l'ACM Special Interest Group in Artificial Intelligence (SIGAI, precedentemente SIGART), l'European Association for AI, la Society for Artificial Intelligence and Simulation of Behaviour (AISB). La Partnership on AI riunisce molte organizzazioni commerciali e non profit preoccupate dell'impatto etico e sociale dell'IA. La rivista *AI Magazine*

dell'AAAI offre articoli e rassegne che trattano molti argomenti e il suo sito web, aaai.org, contiene notizie, tutorial e informazioni di base.

I lavori più recenti appaiono negli atti delle più importanti conferenze sull'IA: l'International Joint Conference on AI (IJCAI), l'European Conference on AI (ECAI) e l'AAAI Conference. L'apprendimento automatico è trattato dall'International Conference on Machine Learning e dal Neural Information Processing Systems (NeurIPS) meeting. Le riviste più importanti per l'IA generica sono *Artificial Intelligence*, *Computational Intelligence*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems* e *Journal of Artificial Intelligence Research*. Ci sono anche molte conferenze e riviste dedicate ad aree specifiche, che tratteremo nei capitoli appropriati.

- 2.1 Agenti e ambienti
- 2.2 Comportarsi correttamente: il concetto di razionalità
- 2.3 La natura degli ambienti
- 2.4 La struttura degli agenti
- 2.5 Riepilogo
Note storiche e bibliografiche

Agenti intelligenti

In cui discutiamo la natura degli agenti, perfetti o no, la diversità degli ambienti e il risultante assortimento di tipi di agenti.

Nel Capitolo 1 abbiamo introdotto il concetto di **agente razionale**, fondamentale per il nostro approccio all'IA. In questo capitolo lo definiremo più concretamente. Come vedremo, il concetto di razionalità si può applicare a una grande varietà di agenti che operano in ogni ambiente immaginabile. Lo scopo di questo libro è applicare tale nozione per sviluppare un piccolo insieme di principî di progettazione per la costruzione di agenti di successo, che possano chiamarsi a buon diritto **intelligenti**.

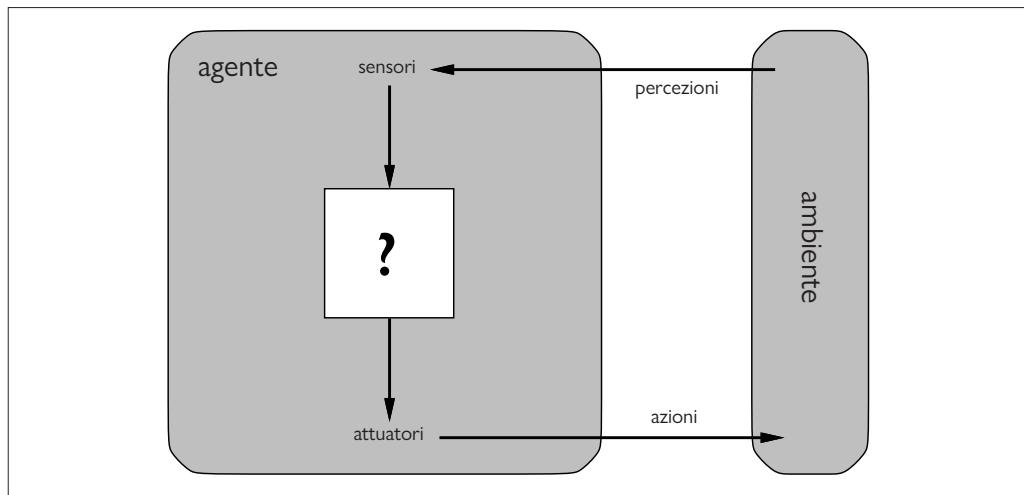
Cominceremo con l'esaminare gli agenti, gli ambienti e le relazioni tra essi. L'osservazione che alcuni agenti si comportano meglio di altri porta naturalmente alla formulazione dell'idea di agente razionale: quello che si comporta nel modo migliore. Quanto bene si può comportare un agente dipende dalla natura degli ambienti; alcuni presentano più difficoltà di altri. Forniremo una semplice suddivisione degli ambienti in categorie e mostreremo come le rispettive caratteristiche influenzano il progetto di agenti adeguati. Descriveremo anche un certo numero di semplici progetti schematici di agente che saranno sviluppati ulteriormente nel resto del libro.

2.1 Agenti e ambienti

Un **agente** è qualsiasi cosa possa essere vista come un sistema che percepisce il suo **ambiente** attraverso dei **sensori** e agisce su di esso mediante **attuatori**. Questa semplice idea è illustrata nella Figura 2.1. Un agente umano possiede come sensori occhi, orecchie e altri organi e può usare come attuatori mani, gambe, tratto vocale e così via. Un agente robotico potrebbe avere telecamere e telemetri a infrarossi per sensori e diversi motori per attuatori. Un agente software riceve come input sensori il contenuto dei file, pacchetti di dati e input umani (attraverso tastiera/mouse/touchscreen/voce) e può intervenire sull'ambiente scrivendo file, inviando pacchetti di rete e visualizzando informazioni o generando suoni. L'ambiente può essere qualsiasi cosa, virtualmente l'universo intero! Ma nella pratica è soltanto la parte dell'universo di cui ci interessa lo stato quando progettiamo l'agente – la parte che influenza ciò che l'agente percepisce e sulla quale influiscono le azioni dell'agente.

Figura 2.1

Gli agenti interagiscono con l'ambiente attraverso sensori e attuatori.

**percezione****sequenza percettiva****funzione agente****programma agente**

Useremo il termine **percezione** (*percept*) per indicare i dati che i sensori di un agente percepiscono. La **sequenza percettiva** (*percept sequence*) di un agente è la storia completa di tutto ciò che esso ha percepito nella sua esistenza. In generale, *la scelta dell'azione di un agente in un qualsiasi istante può dipendere dalla conoscenza integrata in esso e dall'intera sequenza percettiva osservata fino a quel momento, ma non da qualcosa che l'agente non abbia percepito*. Specificando l'azione prescelta dall'agente per ogni possibile sequenza percettiva, abbiamo descritto l'agente in modo completo. In termini matematici diciamo quindi che il comportamento di un agente è descritto dalla **funzione agente**, che descrive la corrispondenza tra una qualsiasi sequenza percettiva e una specifica azione.

Possiamo immaginare di rappresentare *in forma di tabella* la funzione agente che descrive un certo agente. Nella maggior parte dei casi questa tabella sarebbe molto grande, di fatto infinita, a meno di non specificare una lunghezza massima delle sequenze percettive che vogliamo prendere in considerazione. Volendo analizzare un agente, in linea di principio è possibile ricostruire la sua tabella provando tutte le possibili sequenze percettive e registrando l'azione prescelta dall'agente come risposta.¹ La tabella, naturalmente, è una descrizione *esterna* dell'agente. *Internamente*, la funzione agente di un agente artificiale sarà implementata da un **programma agente**. È importante tenere questi due concetti ben distinti: la funzione agente è una descrizione matematica astratta; il programma agente è una sua implementazione concreta, in esecuzione all'interno di un sistema fisico.

Per illustrare queste idee faremo ricorso a un esempio molto semplice: il mondo dell'aspirapolvere, costituito da un agente aspirapolvere robotizzato in un mondo costituito da riquadri che possono essere sporchi o puliti. Nella Figura 2.2 è mostrata una configurazione con due soli riquadri: *A* e *B*. L'agente aspirapolvere percepisce in quale riquadro si trova e se tale riquadro è sporco; parte nel riquadro *A* e le azioni che può compiere sono muoversi a sinistra, a destra, aspirare lo sporco o non fare nulla.² Una funzione agente molto semplice è: “se il riquadro corrente è sporco, aspira, altrimenti muoviti nell'altro riquadro”. La Figura

¹ Se l'agente applica un certo grado di casualità nella scelta delle azioni, potremmo dover provare ogni sequenza più volte per ricostruire la probabilità di ogni scelta. Si potrebbe pensare che agire casualmente sia alquanto stupido, ma più avanti in questo capitolo mostreremo che può essere un comportamento molto intelligente.

² In un robot reale difficilmente sarebbero disponibili azioni come “vai a sinistra” e “vai a destra”, al loro posto ci sarebbero azioni come “fai girare le ruote in avanti” e “fai girare le ruote all’indietro”. In questo esempio abbiamo scelto azioni che risultassero più facili da seguire durante la lettura del testo, e non per la loro facilità di implementazione in un robot reale.

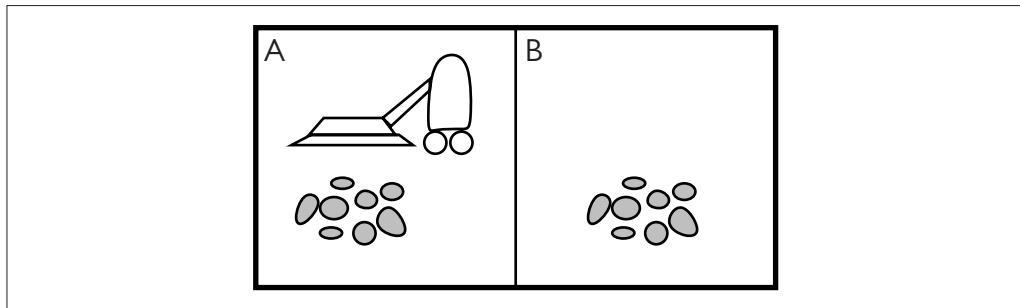


Figura 2.2 Un mondo dell'aspirapolvere con due sole posizioni. Ogni posizione può essere pulita o sporca, e l'agente può andare a sinistra o a destra e può pulire il riquadro in cui si trova. Versioni diverse dell'aspirapolvere consentirebbero regole differenti riguardo ciò che l'agente è in grado di percepire, il fatto che le sue azioni abbiano sempre successo oppure no, e così via.

Sequenza percettiva	Azione
[A, Pulito]	Destra
[A, Sporco]	Aspira
[B, Pulito]	Sinistra
[B, Sporco]	Aspira
[A, Pulito], [A, Pulito]	Destra
[A, Pulito], [A, Sporco]	Aspira
:	:
[A, Pulito], [A, Pulito], [A, Pulito]	Destra
[A, Pulito], [A, Pulito], [A, Sporco]	Aspira
:	:

Figura 2.3 Tabulazione parziale di una semplice funzione agente per il mondo dell'aspirapolvere mostrato nella Figura 2.2. L'agente pulisce il riquadro corrente se è sporco, altrimenti va nell'altro riquadro. Notate che la tabella ha dimensioni illimitate, a meno che non vi sia un limite sulla lunghezza delle possibili sequenze percettive.

2.3 ne mostra una tabulazione parziale: forniremo più avanti in questo capitolo, nella Figura 2.8, un programma agente che la implementa.

Osservando la Figura 2.3, vediamo che si possono definire vari agenti del mondo aspirapolvere semplicemente riempiendo la colonna di destra in modi diversi. La domanda che sorge naturalmente, allora, è questa: *qual è il modo più corretto di progettare la tabella?* In altre parole, che cosa rende un agente buono o cattivo, intelligente o stupido? Risponderemo a queste domande nel prossimo paragrafo.

Prima di concludere questo paragrafo, desideriamo porre l'accento sul fatto che il concetto di agente è solo uno strumento adatto all'analisi di sistemi, e non una caratterizzazione assoluta che divide il mondo in agenti e non-agenti. Si potrebbe anche dire che una calcolatrice portatile è un agente che decide di intraprendere l'azione di visualizzare “4” sul display quando gli viene fornita la sequenza percettiva “2 + 2 =”, ma una siffatta analisi servirebbe ben poco a comprendere il suo funzionamento. In un certo senso, tutte le aree dell'ingegneria possono essere viste come la progettazione di artefatti che interagiscono con il mondo; l'IA opera all'estremità più interessante dello spettro (secondo la nostra opinione), dove gli artefatti dispongono di risorse computazionali significative e l'ambiente richiede di prendere decisioni non banali.



2.2 Comportarsi correttamente: il concetto di razionalità

agente razionale

Un **agente razionale** è un agente che fa la cosa giusta. È ovvio che è meglio fare la cosa giusta che quella sbagliata, ma che cosa significa precisamente?

consequenzialismo

2.2.1 Misure di prestazione

In filosofia morale sono state sviluppate diverse nozioni di “cosa giusta”, ma l’IA in generale è rimasta fedele alla nozione di **consequenzialismo**: valutiamo il comportamento di un agente considerandone le *conseguenze*. Quando un agente viene inserito in un ambiente, genera una sequenza di azioni in base alle percezioni che riceve. Questa sequenza di azioni porta l’ambiente ad attraversare una sequenza di stati: se tale sequenza è desiderabile, significa che l’agente si è comportato bene. Questa nozione di desiderabilità è catturata da una **misura di prestazione** che valuta una sequenza di stati dell’ambiente.

misura di prestazione

Gli umani hanno desideri e preferenze personali, perciò la nozione di razionalità applicata agli umani riguarda il successo nella scelta di azioni che producano sequenze di stati dell’ambiente desiderabili *dal loro punto di vista*. Le macchine, invece, *non* hanno desideri e preferenze personali; la misura di prestazione, almeno inizialmente, sta nella mente del loro progettista, o nella mente degli utenti a cui la macchina è destinata. Vedremo che in alcuni progetti di agenti vi è una rappresentazione esplicita di (una versione) della misura di prestazione, mentre in altri la misura di prestazione è del tutto implicita: l’agente può fare la cosa giusta, ma non sa perché.

Ricordando l’invito di Norbert Wiener ad assicurarci che “lo scopo incorporato nella macchina sia proprio quello che desideriamo” (cfr. la fine del Paragrafo 1.5 nel Capitolo 1), notiamo che può essere assai difficile formulare correttamente una misura di prestazione. Consideriamo, per esempio, l’agente aspirapolvere del paragrafo precedente. Come misura di prestazione potremmo proporre la quantità di sporco aspirato in un singolo lasso di tempo di otto ore. Nel caso di un agente razionale, naturalmente, si ottiene sempre quello che si chiede: l’aspirapolvere potrebbe massimizzare la misura pulendo un po’ di sporco, quindi rovesciandolo nuovamente sul pavimento per poi aspirarlo ancora e così via. Una misura di prestazione più adeguata potrebbe fare riferimento al grado di pulizia del pavimento: per esempio, si potrebbe assegnare un punto per ogni riquadro pulito in ogni passo temporale (magari assegnando una penalità per il consumo di elettricità e il rumore generato). *Come regola generale, è meglio progettare le misure di prestazione in base all’effetto che si desidera ottenere sull’ambiente piuttosto che su come si pensa che debba comportarsi l’agente.*



Anche quando si evitano le trappole più evidenti, rimangono alcuni problemi intricati da sciogliere. Per esempio, il concetto di “pulizia del pavimento” che abbiamo espresso nel paragrafo precedente è basato sulla pulizia media in un lasso di tempo. Eppure la stessa pulizia media può essere ottenuta da due agenti ben diversi: uno che pulisce in modo mediocre ma costante, l’altro che pulisce con energia prendendosi però delle lunghe pause. Determinare quale delle due soluzioni sia preferibile può sembrare un problema che riguarda solo bidelli e portinai, ma è effettivamente un profondo dilemma filosofico con vaste implicazioni. È meglio una vita smodata fatta di altissimi picchi e profonde cadute, o una sicura ma monotona? È meglio un’economia in cui tutti vivono in moderata povertà, o una in cui alcuni sono ricchissimi e altri molto poveri? Lasceremo le risposte a queste domande come esercizio per i lettori più diligenti.

Nella maggior parte di questo libro assumeremo che la misura di prestazione possa essere specificata correttamente. Tuttavia, per i motivi appena descritti, dobbiamo accettare la possibilità che lo scopo incorporato nella macchina sia errato – è il problema di Re Mida descritto alla fine del Paragrafo 1.5. Inoltre, quando progettiamo un software destinato a essere

distribuito a diversi utenti, non possiamo prevedere le esatte preferenze di ogni singolo utente; potrebbe quindi essere necessario costruire agenti che riflettano l'incertezza iniziale circa la vera misura di prestazione e in grado di apprenderla con il passare del tempo. Tali agenti sono descritti nei Capitoli 16, 18 del presente volume e 22 del volume 2.

2.2.2 Razionalità

In un dato momento, ciò che è razionale dipende da quattro fattori:

- la misura di prestazione che definisce il criterio del successo;
- la conoscenza pregressa dell'ambiente da parte dell'agente;
- le azioni che l'agente può effettuare;
- la sequenza percettiva dell'agente fino all'istante corrente.

Questo porta alla seguente **definizione di agente razionale**:

per ogni possibile sequenza di percezioni, un agente razionale dovrebbe scegliere un'azione che massimizzi il valore atteso della sua misura di prestazione, date le informazioni fornite dalla sequenza percettiva e da ogni ulteriore conoscenza dell'agente.

**definizione di
agente razionale**



Consideriamo il semplice agente aspirapolvere che pulisce un riquadro se questo è sporco, muovendosi nell'altro in caso contrario; la sua tabella è rappresentata nella Figura 2.3. Questo agente è razionale? Dipende! Dobbiamo prima di tutto dire qual è la sua misura di prestazione, che cosa sa dell'ambiente, quali sensori e attuatori possiede. Supponiamo che:

- la misura di prestazione assegna un punto per ogni riquadro pulito all'inizio di ogni passo temporale, per una “vita” dell'agente di 1000 passi;
- sia nota *a priori* la “geografia” dell'ambiente (Figura 2.2) ma non la posizione iniziale dell'agente, né la distribuzione dello sporco. I riquadri puliti rimangono tali, mentre l'aspirazione dello sporco pulisce il riquadro corrente. Le azioni *Destra* e *Sinistra* muovono l'agente di un riquadro nelle corrispondenti direzioni, tranne quando ciò porterebbe l'agente fuori dall'ambiente, nel qual caso l'agente non si muove;
- le sole azioni disponibili sono *Destra*, *Sinistra* e *Aspira*;
- l'agente percepisce correttamente la propria posizione e se il riquadro corrente è sporco o meno.

Date queste condizioni l'agente è effettivamente razionale; le sue prestazioni sono buone almeno quanto quelle di qualsiasi altro agente.

Si vede chiaramente che lo stesso agente, date altre condizioni, non sarebbe più razionale. Per esempio, una volta pulito tutto lo sporco quest'agente continua a muoversi avanti e indietro; se la misura di prestazione prevedesse una penalità di un punto per ogni movimento, l'agente se la caverebbe maluccio. In questo caso sarebbe meglio non fare nulla, una volta assicurarsi che tutti i riquadri siano puliti. D'altra parte, se i riquadri puliti potessero sporcarsi nuovamente, l'agente dovrebbe controllare ogni tanto, e nel caso ripulire ove necessario. Se la geografia dell'ambiente fosse sconosciuta, l'agente dovrebbe **esplorarlo**. L'Esercizio 2.VACR vi chiederà di progettare agenti adatti a tutti questi casi.

2.2.3 Onniscienza, apprendimento e autonomia

Dobbiamo distinguere accuratamente il concetto di razionalità e quello di **onniscienza**. Un agente onnisciente conosce il risultato *effettivo* delle sue azioni e può agire di conseguenza, ma nella realtà l'onniscienza è impossibile. Consideriamo il seguente esempio: sto camminando un bel giorno lungo gli Champs Elysées e vedo un vecchio amico dall'altra parte della strada. Non ci sono automobili vicino e non ho nulla da fare per cui, razionalmente, comincio ad attraversare la strada. Nel frattempo, a undicimila metri di quota, il portellone di un aereo

onniscienza

si stacca³ e vengo spiaccicato prima di arrivare dall'altra parte della strada. È stato irrazionale, da parte mia, decidere di attraversare? È molto improbabile che il mio necrologio reciti: “Un idiota cerca di attraversare gli Champs Elysées”.

Quest'esempio dimostra che razionalità non significa perfezione. Infatti la razionalità massimizza il risultato *atteso*, mentre la perfezione massimizza quello *reale*. Non richiedere prestazioni perfette non è solo una questione di generosità verso gli agenti: per pretendere che un agente faccia quella che solo a posteriori si può rivelare la migliore azione, dovremmo utilizzare sfere di cristallo o macchine del tempo.

La nostra definizione di razionalità non richiede quindi l'onniscienza, perché la scelta razionale dipende solo dalla sequenza percettiva *fino al momento corrente*. Dobbiamo anche assicurarci di non aver inavvertitamente permesso all'agente di intraprendere con risolutezza attività poco intelligenti. Per esempio, se un agente non guarda in entrambe le direzioni prima di attraversare una strada trafficata, la sua sequenza percettiva non gli dirà che c'è un TIR che si avvicina a gran velocità. Date queste premesse, secondo la nostra definizione sarebbe razionale attraversare la strada in questo momento? Nient'affatto! Prima di tutto, con una sequenza percettiva così scarsa di informazioni il rischio di incidenti sarebbe troppo grande. In secondo luogo, un agente razionale dovrebbe scegliere l'azione “guarda” prima di incominciare l'attraversamento, perché così facendo potrebbe massimizzare la prestazione attesa. Intraprendere azioni *mirate a modificare le percezioni future* – ciò che viene talvolta chiamato, con un termine inglese, **information gathering** (raccolta di informazioni) – è una parte importante della razionalità. Un secondo esempio di information gathering è rappresentato dall'**esplorazione** che dev'essere intrapresa da un agente aspirapolvere posto in un ambiente inizialmente sconosciuto.

information
gathering
esplorazione

apprendimento

La nostra definizione richiede che un agente razionale non si limiti a raccogliere informazioni, ma sia anche in grado di **apprendere** il più possibile sulla base delle proprie percezioni. La sua configurazione iniziale potrebbe riflettere una conoscenza pregressa sull'ambiente, che però può modificarsi e aumentare a mano a mano che l'agente accumula esperienza. Ci sono casi limite nei quali l'ambiente è completamente conosciuto a priori e completamente predicibile: in tali situazioni l'agente non ha bisogno di percepire o apprendere, può semplicemente agire nel modo corretto.

Naturalmente, il comportamento di agenti simili è molto fragile. Consideriamo l'umile scarabeo stercorario: dopo aver scavato la sua tana e aver deposto le uova, l'insetto recupera una pallina di sterco dalle vicinanze per bloccarne l'ingresso. Se la palla di sterco viene rimossa dalle sue zampe durante il tragitto, lo scarabeo continua a camminare e compie gli stessi gesti come se stesse tappando la tana con la palla inesistente, non accorgendosi mai della sua scomparsa. L'evoluzione ha selezionato il comportamento dello scarabeo basandosi su un assunto, e quando questo è violato, ne consegue un comportamento erroneo. La vespa sphex è leggermente più intelligente. Le femmine scavano una tana, quindi escono e vanno a punger un bruco che trascinano fino alla tana stessa. Una volta giunte entrano nuovamente per controllare che tutto sia a posto, quindi trascinano dentro il bruco e finalmente depongono le uova. Fin qui tutto bene, ma se un entomologo sposta il bruco qualche centimetro più in là mentre la vespa sta facendo il controllo, questa ritornerà indietro fino al passo “trascina il bruco” del suo piano, dopodiché lo riprenderà senza modifiche, ricontralando la tana, anche dopo che il bruco è stato spostato dozzine di volte. La vespa sphex non riesce a capire che il suo piano istintivo sta fallendo, e così non può cambiarlo.

Quando un agente si appoggia alla conoscenza pregressa fornita dal progettista invece che alle proprie percezioni e ai suoi processi di apprendimento, diciamo che manca di **auto-**

³ Cfr. N. Henderson, “New door latches urged for Boeing 747 jumbo jets” *Washington Post*, 24 agosto 1989.

nomia. Un agente razionale dovrebbe essere autonomo e apprendere il più possibile per compensare la presenza di conoscenza parziale o erronea. Per esempio, un agente aspirapolvere che apprende come predire quando e dove apparirà il nuovo sporco si comporterà meglio degli altri. Nella pratica, raramente si richiede che un agente sia completamente autonomo fin dall'inizio: finché la sua esperienza è limitata o nulla, un simile agente sarebbe costretto ad agire casualmente, a meno di non ricevere aiuto dal suo progettista. Proprio come l'evoluzione ha fornito agli animali riflessi innati per sopravvivere abbastanza a lungo da apprendere come comportarsi, è ragionevole fornire a un agente intelligente artificiale, oltre all'abilità di apprendere, anche un po' di conoscenza iniziale. Dopo aver accumulato una sufficiente esperienza in un dato ambiente, il comportamento dell'agente razionale può diventare a tutti gli effetti *indipendente* dalla conoscenza pregressa. Incorporare l'apprendimento nel suo progetto, quindi, permette di sviluppare un agente razionale capace di operare efficacemente in una grande varietà di ambienti differenti.

autonomia

2.3 La natura degli ambienti

Ora che abbiamo una definizione di razionalità, siamo quasi pronti a occuparci della creazione di agenti razionali. Prima, però, dobbiamo considerare gli **ambienti operativi**, che sono essenzialmente i “problemi” di cui gli agenti razionali rappresentano le “soluzioni”. Cominceremo a vedere, con l'aiuto di numerosi esempi, come definire un ambiente. Mostreremo poi che ci sono molte tipologie di ambienti, le quali influenzano direttamente la progettazione del programma agente più appropriato.

ambiente operativo

2.3.1 Specificare un ambiente

Quando abbiamo discusso la razionalità del nostro semplice agente aspirapolvere, abbiamo dovuto specificare la misura di prestazione, l'ambiente esterno e gli attuatori e i sensori dell'agente. Tutto ciò può essere raggruppato nel termine **ambiente operativo** (*task environment*). Se vi piacciono gli acronimi, potete parlare anche di descrizione **PEAS** (*Performance, Environment, Actuators, Sensors*). Quando si progetta un agente, il primo passo dovrebbe sempre corrispondere alla specifica dell'ambiente operativo, la più ricca possibile.

PEAS

Il mondo dell'aspirapolvere era un esempio semplice. Consideriamone ora uno più complesso: un autista automatico per taxi. La Figura 2.4 presenta la descrizione PEAS dell'ambiente operativo del taxi. Ne discuteremo ogni elemento nei prossimi paragrafi.

misura di prestazione

Prima di tutto, a quale **misura di prestazione** dovrebbe aspirare il nostro autista automatico? Gli aspetti desiderabili includono: arrivare alla destinazione corretta; minimizzare il consumo di carburante e l'usura; minimizzare la durata temporale del viaggio e/o il suo costo; minimizzare le violazioni al codice della strada e il disturbo per gli altri guidatori; massimizzare la sicurezza e il comfort dei passeggeri; massimizzare i profitti. Alcuni di questi obiettivi sono chiaramente in contrasto, per cui occorrerà trovare una qualche forma di compromesso.

tipo di agente	misura di prestazione	ambiente	attuatori	sensori
guidatore di taxi	sicuro, veloce, ligo alla legge, viaggio confortevole, massimizza i profitti, minimizza l'impatto su altri utenti della strada	strada, altri veicoli nel traffico, polizia, pedoni, clienti, tempo atmosferico	sterzo, acceleratore, freni, frecce, clacson, schermo, voce	telecamere, radar, tachimetro, GPS, sensori del motore, accelerometro, microfoni, touchscreen

Figura 2.4
Descrizione PEAS dell'ambiente operativo di un autista di taxi automatizzato.

ambiente	Il punto successivo è: qual è l' ambiente in cui il taxi dovrà operare? Ogni tassista dev'essere in grado di muoversi su una varietà di strade, dagli sterrati di campagna alle strade urbane, fino alle autostrade a più corsie. Le strade contengono altri veicoli, pedoni, animali randagi, lavori in corso, macchine della polizia, pozzae e buche nell'asfalto. Il taxi deve anche interagire con i passeggeri, potenziali e non. Ci sono anche scelte aggiuntive: il taxi potrebbe operare in California, dove c'è raramente la neve, oppure in Alaska, dove è raro che non ci sia. Potremmo accontentarci di un taxi che guida a destra, o potremmo desiderarne uno abbastanza flessibile da tenere la sinistra quando si trova in Inghilterra o Giappone. Naturalmente, restringendo l'ambiente la progettazione diventa più semplice.
attuatore	Gli attuatori disponibili all'autista automatico includeranno quelli usati da un autista umano: acceleratore, sterzo e freno. Oltre a questi, sarà necessario uno schermo o un sistema di sintesi vocale per comunicare con i passeggeri, e forse anche qualche dispositivo per comunicare con gli altri veicoli, in modo più o meno cortese.
sensore	I sensori di base del taxi comprenderanno almeno una o più telecamere per poter vedere la strada, con eventuali telemetri laser o sensori a ultrasuoni per rilevare le distanze da altri veicoli e ostacoli. Per evitare multe per eccesso di velocità servirà un tachimetro e per un controllo appropriato del veicolo, specialmente in curva, servirà un accelerometro. Per conoscere lo stato meccanico del veicolo sarà anche necessaria la presenza dei consueti sensori per motore, carburante e sistema elettrico. Come per molti autisti umani, sarebbe utile un sistema di posizionamento globale satellitare (GPS), in modo da non perdersi. Infine, il passeggero avrà bisogno di un touchscreen o di un sistema di input vocale per indicare la sua destinazione.
agente software softbot	Nella Figura 2.5 abbiamo abbozzato gli elementi base PEAS per diversi altri tipi di agente. Ulteriori esempi sono riportati nell'Esercizio 2.PEAS. Tra gli esempi vi sono ambienti fisici e virtuali. Notate che alcuni ambienti virtuali possono essere complessi quanto quelli "reali": per esempio, un agente software (detto anche "software robot", o softbot) che opera su siti web di aste e di commercio al dettaglio deve avere a che fare con milioni di altri utenti e miliardi di oggetti, alcuni con immagini reali.

Figura 2.5
Esempi di tipi di agente e loro descrizioni PEAS.

tipo di agente	misura di prestazione	ambiente	attuatori	sensori
sistema di diagnosi medica	paziente sano, costi ridotti	paziente, ospedale, staff medico	schermo per visualizzare domande, test, diagnosi, trattamenti	touchscreen/input vocale per l'inserimento dei sintomi e dei risultati
sistema di analisi di immagini satellitari	categorizzazione corretta di oggetti, terreni	satellite orbitante, collegamento dati verso terra, tempo atmosferico	visualizzazione della categorizzazione della scena	telecamera digitale ad alta risoluzione
robot selezionatore di parti meccaniche	percentuale di pezzi inseriti nei contenitori giusti	nastro trasportatore con parti meccaniche, contenitori	braccio meccanico con manipolatore	telecamera, sensori tattili e di posizionamento del braccio meccanico
controllore industriale per una raffineria	purezza, volume della produzione, sicurezza	raffineria, materie prime, operai specializzati	valvole, pompe, elementi di riscaldamento, agitatori, schermi	sensori di temperatura e pressione, di flusso, sensori chimici
tutor interattivo per lo studio dell'inglese	risultati del test dello studente	insieme di studenti, istituto erogatore dei test	visualizzazione di esercizi, feedback, discorsi	input da tastiera, input vocale

2.3.2 Proprietà degli ambienti operativi

La varietà di ambienti operativi nell'IA è naturalmente molto vasta. È comunque possibile identificare un numero relativamente piccolo di dimensioni in base a cui suddividerli in categorie. Queste dimensioni determinano in gran parte la progettazione di agenti appropriati e l'applicabilità delle principali tecniche per la loro implementazione. Prima di tutto elencheremo le dimensioni, in seguito analizzeremo diversi ambienti operativi esemplificativi. Le definizioni seguenti sono informali: nei capitoli successivi ne forniremo di più precise, insieme ad altri esempi di ogni tipo di ambiente.

- **Completemente osservabile/parzialmente osservabile:** se i sensori di un agente gli danno accesso allo stato completo dell'ambiente in ogni momento, allora diciamo che l'ambiente operativo è completamente osservabile. In effetti, perché un ambiente operativo sia completamente osservabile basta che i sensori dell'agente misurino tutti gli aspetti che sono *rilevanti* per la scelta dell'azione; la rilevanza a sua volta dipende dalla misura di prestazione. Gli ambienti completamente osservabili sono comodi, perché l'agente non deve memorizzare uno stato interno per tenere traccia del mondo. Un ambiente potrebbe essere parzialmente osservabile a causa di sensori inaccurati o per la presenza di rumore, o semplicemente perché una parte dei dati non viene rilevata dai sensori: per esempio, un agente aspirapolvere con un sensore locale di rilevazione della sporcizia non potrà sapere se ci sono altri riquadri sporchi, e un autista automatico per taxi non potrà in ogni caso sapere le intenzioni degli altri guidatori. Se l'agente non dispone di sensori, l'ambiente è **inosservabile**. Si potrebbe pensare che in questi casi la condizione dell'agente sia senza speranza, invece, come vedremo nel Capitolo 4, i suoi obiettivi potrebbero risultare ancora raggiungibili, talvolta con certezza.

completamente
osservabile
parzialmente
osservabile

inosservabile

- **Agente singolo/multiagente:** la distinzione tra ambienti ad agente singolo e multiagente può sembrare piuttosto semplice. Per esempio, un agente che da solo risolve un cruciverba opera in un ambiente ad agente singolo, laddove un agente che gioca a scacchi si trova in un ambiente a due agenti. Tuttavia la questione è sottile. Innanzitutto, abbiamo descritto in che modo un'entità *può* essere vista come agente, ma non abbiamo spiegato quali entità *devono* essere considerate tali. Un agente *A* (l'autista del taxi, per esempio) deve considerare l'oggetto *B* (un altro veicolo) come un agente, oppure lo può trattare come un semplice oggetto che si comporta secondo le leggi della fisica, simile alle onde sul bagnasciuga o a foglie spinte dal vento? La distinzione chiave è se si può descrivere il comportamento di *B* come il tentativo di massimizzare una misura di prestazione il cui valore dipende dal comportamento dell'agente *A*. Per esempio, negli scacchi l'avversario *B* sta cercando di massimizzare una misura di prestazione che, per le regole degli scacchi, minimizza quella dell'agente *A*. Gli scacchi, quindi, sono un ambiente multiagente **competitivo**. Nell'ambiente dei taxi, al contrario, evitare gli incidenti massimizza la misura di prestazioni di tutti gli agenti, per cui possiamo dire che quello è un ambiente multiagente **parzialmente cooperativo**. È anche parzialmente competitivo perché, tra le altre cose, una sola macchina per volta può occupare un parcheggio libero. I problemi di progettazione che sorgono negli ambienti multiagente sono spesso molto differenti da quelli ad agente singolo: per esempio, spesso nei primi può emergere come comportamento razionale la comunicazione. In alcuni ambienti competitivi, il comportamento randomizzato è razionale perché permette di evitare la predicitività delle azioni.

agente singolo
multiagente

competitivo

cooperativo

- **Deterministico/non deterministico:** se lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseguita dall'agente (o dagli agenti), allora si può dire che l'ambiente è deterministico; in caso contrario si dice che è non deterministico. In via di principio, un agente non si deve preoccupare dell'incertezza se si trova in un ambiente completamente osservabile e deterministico. Tuttavia, se l'ambiente è solo

deterministico

stocastico

parzialmente osservabile potrebbe *sembrare* non deterministico. La maggior parte delle situazioni reali è talmente complessa che risulta impossibile tenere traccia di tutti gli aspetti non osservati, aspetti non osservati; per scopi pratici, devono essere considerate non deterministiche. In questi termini, guidare un taxi è chiaramente una situazione non deterministica, perché nessuno può prevedere esattamente l'andamento del traffico; inoltre le forature sono quasi sempre inaspettate e i guasti del motore si possono verificare senza preavviso. Il mondo dell'aspirapolvere come l'abbiamo descritto è deterministico, ma si potrebbero facilmente descrivere delle varianti che includono elementi non deterministicici, come sporco che appare casualmente o un meccanismo di aspirazione poco affidabile (Esercizio 2.VFIN). Un'ultima nota: alcuni utilizzano il termine **stocastico** come sinonimo di “non deterministico”, mentre noi abbiamo preferito distinguerli; diciamo che un modello dell’ambiente è stocastico se è esplicitamente associato a probabilità (per esempio “c’è una probabilità del 25% che domani piova”) e “non deterministico” se le varie possibilità sono elencate senza essere quantificate (per esempio “c’è la possibilità che domani piova”).

**episodico
sequenziale**

- **Episodico/sequenziale:** in un ambiente operativo episodico, l’esperienza dell’agente è divisa in episodi atomici. In ogni episodio l’agente riceve una percezione e poi esegue una singola azione. L’aspetto fondamentale è che un episodio non dipende dalle azioni intraprese in quelli precedenti. Molte attività di classificazione sono episodiche. Per esempio, un agente che deve identificare i pezzi difettosi in una catena di montaggio basa ogni decisione solamente sul pezzo esaminato in quel momento, indipendentemente dalle decisioni prese in precedenza; inoltre, la decisione corrente non renderà più o meno probabile che il pezzo successivo sia difettoso. Negli ambienti sequenziali, al contrario, ogni decisione può influenzare tutte quelle successive.⁴ Gli scacchi e la guida dei taxi sono sequenziali: in entrambi i casi le azioni a breve termine possono avere conseguenze a lungo termine. Gli ambienti episodici sono molto più semplici di quelli sequenziali, perché l’agente non deve “pensare avanti”.

**statico
dinamico****semidinamico****discreto
continuo**

- **Statico/dinamico:** se l’ambiente può cambiare mentre un agente sta decidendo come agire, allora diciamo che è dinamico per quell’agente; in caso contrario diciamo che è statico. Gli ambienti statici sono più facili da trattare perché l’agente non deve continuamente osservare il mondo mentre decide l’azione successiva e non si deve preoccupare del passaggio del tempo. Gli ambienti dinamici, invece, chiedono continuamente all’agente quello che vuole fare; se questo non risponde in tempo, è come se avesse deciso di non fare nulla. Se l’ambiente stesso non cambia al passare del tempo, ma la valutazione della prestazione dell’agente sì, allora diciamo che l’ambiente è **semidinamico**. Guidare un taxi è chiaramente dinamico: le altre macchine e il taxi stesso continuano a muoversi mentre l’algoritmo di guida pondera l’azione successiva. Gli scacchi sono semidinamici se giocati con l’orologio. I cruciverba sono statici.

- **Discreto/continuo:** la distinzione tra discreto e continuo si applica allo *stato* dell’ambiente, al modo in cui è gestito il *tempo*, alle *percezioni* e *azioni* dell’agente. Per esempio, la scacchiera ha un numero finito di stati distinti (se si esclude il tempo). Gli scacchi hanno anche un insieme discreto di percezioni e azioni. La guida di un taxi è un problema con stato e tempo continui: la velocità e la posizione del taxi e degli altri veicoli cambiano con continuità al passare del tempo. In questo caso sono continue anche le azioni (pensiamo per esempio all’angolo di sterzo delle ruote). L’input proveniente da una telecamera digitale, strettamente parlando, è discreto, ma tipicamente si considera che rappresenti intensità e posizioni che variano con continuità.

⁴ La parola “sequenziale” è anche usata nell’informatica in contrapposizione a “parallelo”. I due significati sono indipendenti.

- **Noto/ignoto:** in termini rigorosi, questa distinzione non si riferisce all'ambiente in sé, ma allo stato di conoscenza dell'agente (o del progettista) circa le "leggi fisiche" dell'ambiente stesso. In un ambiente noto, sono conosciuti i risultati (o le corrispondenti probabilità, se l'ambiente è stocastico) per tutte le azioni. Ovviamente, se l'ambiente è ignoto, l'agente dovrà apprendere come funziona per poter prendere buone decisioni. La distinzione tra ambienti noti e ignoti non è identica a quella tra ambienti completamente osservabili e parzialmente osservabili: è infatti possibile che un ambiente *noto* sia *parzialmente osservabile*; per esempio, nei giochi a carte di solitario, il giocatore conosce le regole ma non può vedere le carte che non sono ancora state girate. Viceversa, un ambiente *ignoto* può essere *completamente osservabile*: in un nuovo videogioco, lo schermo potrebbe mostrare l'intero stato del gioco, ma il giocatore non conosce l'effetto dei pulsanti finché non li prova. Come si è osservato nel Paragrafo 2.2.1, anche la misura di prestazione potrebbe essere ignota, o perché il progettista non è sicuro di come scriverla correttamente, o perché l'utente finale – le cui preferenze contano – non è noto. Per esempio, un autista di taxi generalmente non sa se un nuovo passeggero preferisce un viaggio tranquillo o veloce, una guida cauta o aggressiva. Un assistente personale virtuale all'inizio non sa nulla circa le preferenze personali del suo proprietario. In questi casi, l'agente può apprendere di più sulla misura di prestazione basandosi su ulteriori interazioni con il progettista o l'utente. Questo, a sua volta, indica che l'ambiente operativo è necessariamente visto come ambiente multiagente.

Il caso più difficile è quello *parzialmente osservabile, multiagente, non deterministico, sequenziale, dinamico, continuo e ignoto*. Guidare un taxi è difficile sotto tutti questi punti di vista, a parte il fatto che l'ambiente del guidatore è per lo più noto. Guidare un'auto a noleggio in un paese estero con territorio non familiare, codice della strada diverso e passeggeri nervosi sarebbe molto più interessante.

La Figura 2.6 elenca le proprietà di diversi ambienti di esempio. Notate che le proprietà non sono sempre sicure al cento per cento. Per esempio, abbiamo indicato l'attività di diagnosi medica ad agente singolo perché non c'è ragione di modellare come agente il processo patologico in atto nel paziente; il sistema di diagnosi medica, però, potrebbe dover gestire pazienti recalcitranti e medici scettici, e questo potrebbe essere rappresentato come un aspetto multiagente. Inoltre, la diagnosi medica è episodica se si immagina di formularne semplicemente una sulla base di una lista di sintomi; il problema diventa sequenziale se l'attività prevede di proporre una serie di test, valutare l'efficacia di una terapia, gestire diversi pazienti e così via.

ambiente operativo	osservabile	agenti	deterministico	episodico	statico	discreto
cruciverba	completamente	singolo	deterministico	sequenziale	statico	discreto
scacchi con orologio	completamente	multi	deterministico	sequenziale	semi	discreto
poker	parzialmente	multi	stocastico	sequenziale	statico	discreto
backgammon	completamente	multi	stocastico	sequenziale	statico	discreto
autista di taxi	parzialmente	multi	stocastico	sequenziale	dinamico	continuo
diagnosi medica	parzialmente	singolo	stocastico	sequenziale	dinamico	continuo
analisi di immagini	completamente	singolo	deterministico	episodico	semi	continuo
robot selezionatore di parti meccaniche	parzialmente	singolo	stocastico	episodico	dinamico	continuo
controllore industriale per una raffineria	parzialmente	singolo	stocastico	sequenziale	dinamico	continuo
tutor interattivo per lo studio dell'inglese	parzialmente	multi	stocastico	sequenziale	dinamico	discreto

Figura 2.6 Esempi di ambienti operativi e loro caratteristiche.

classe di ambienti**programma agente****architettura**

Non abbiamo inserito una colonna “noto/ignoto” perché, come abbiamo spiegato precedentemente, questa non è strettamente una proprietà dell’ambiente. Per alcuni ambienti, come gli scacchi e il poker, è abbastanza facile fornire all’agente la piena conoscenza delle regole, ma rimane interessante considerare come un agente potrebbe imparare a giocare questi giochi senza disporre di tale conoscenza.

Il repository di codice associato a questo libro (aima.cs.berkeley.edu) include l’implementazione di un certo numero di ambienti, insieme a un simulatore di ambienti di uso generale per valutare la prestazione di un agente. Questi esperimenti spesso non vengono eseguiti per un singolo ambiente, ma per più ambienti di una **classe di ambienti**. Per valutare un autista di taxi immerso nel traffico, per esempio, vorremo eseguire molte simulazioni variando il traffico, le condizioni di luce e quelle meteorologiche. Quello che ci interessa sono le prestazioni medie dell’agente su tutta la classe di ambienti.

2.4 La struttura degli agenti

Fin qui, parlando degli agenti, ci siamo limitati a descrivere il loro *comportamento*, ovvero l’azione eseguita in corrispondenza di una data sequenza di percezioni. Ora dovremo fare un passo avanti e cominciare a descrivere il loro funzionamento interno. Il compito dell’IA è progettare il **programma agente** che implementa la funzione agente – che fa corrispondere le percezioni alle azioni. Diamo per scontato che questo programma sarà eseguito da un dispositivo computazionale dotato di sensori e attuatori fisici; questa prende il nome di **architettura agente**:

$$\text{agente} = \text{architettura} + \text{programma}.$$

Naturalmente, il programma prescelto dovrà essere appropriato per l’architettura: per esempio, se il programma prevede azioni come *Cammina*, l’architettura dovrà essere fornita di gambe. L’architettura potrebbe essere costituita da un semplice PC di uso quotidiano, o da un veicolo robotico dotato di vari computer, telecamere e altri sensori. In generale l’architettura si occupa di rendere le percezioni disponibili al programma, eseguire il programma stesso e passare le azioni da esso prescelte agli attuatori a mano a mano che vengono generate.

2.4.1 Programmi agente

I programmi agente che progetteremo nel corso del libro hanno tutti la stessa struttura: prendono come input la percezione corrente dei sensori e restituiscono un’azione agli attuatori.⁵ Notate la differenza tra il programma agente, che prende come input solamente la percezione corrente, e la funzione agente, che potrebbe dipendere dall’intera storia delle percezioni. Il programma agente è costretto a basarsi sulla sola percezione corrente perché l’ambiente non può fornirgli nulla di più; se le sue azioni devono dipendere dalla sequenza percettiva precedente, l’agente stesso dovrà preoccuparsi di memorizzarla.

Descriveremo i programmi agente nel semplice pseudocodice definito nell’Appendice B (il repository online di codice contiene implementazioni in vari linguaggi di programmazione reali). Per esempio, la Figura 2.7 mostra un programma agente abbastanza banale che tiene traccia della sequenza percettiva e poi la usa per selezionare l’azione da intraprendere in una tabella. La tabella (di cui è fornito un esempio per il mondo dell’aspirapolvere nella Figura 2.3) rappresenta in modo esplicito la funzione agente implementata dal programma

⁵ Ci sono altre possibili scelte per lo schema di un programma agente; ad esempio questo potrebbe essere realizzato mediante **coroutine** in esecuzione asincrona insieme all’ambiente. Ogni coroutine avrebbe porte di input e output e consisterebbe in un ciclo infinito che legge dalla porta di input e scrive azioni su quella di output.

```
function AGENTE-CON-TABELLA(percezione) returns un'azione
persistent: percezioni, una sequenza inizialmente vuota
tabella, una tabella di azioni, indicizzata per sequenze percettive, completamente
specificata dall'inizio

aggiungi percezione alla fine di percezioni
azione  $\leftarrow$  LOOKUP(percezioni, tabella)
return azione
```

Figura 2.7
Il programma AGENTE-CON-TABELLA viene invocato per ogni nuova percezione e restituisce ogni volta l'azione da eseguire. Mantiene in memoria la sequenza percettiva completa.

agente. Per costruire un agente razionale con questa tecnica, i progettisti devono costruire una tabella che contenga l'azione appropriata per ogni possibile sequenza percettiva.

È utile comprendere subito perché l'approccio basato su tabelle esplicite sia destinato al fallimento. Sia P l'insieme di possibili percezioni e T la durata della vita dell'agente (ovvero il numero totale di percezioni che riceverà). La tabella dovrà contenere $\sum_{t=1}^T |P|^t$ righe. Considerate il caso del taxi automatizzato: l'input visuale di una singola telecamera (le auto autonome ne hanno tipicamente 8) corrisponde a un flusso di informazioni di circa 70 mega-byte al secondo (30 fotogrammi a una risoluzione di 1080×720 pixel con 24 bit di profondità di colore). Per un'ora di guida, questo significa che la tabella dovrà avere $10^{600.000.000.000}$ righe. Anche la tabella per il gioco degli scacchi – un frammento molto piccolo del mondo reale, che varia secondo modalità ben conosciute – ha almeno 10^{150} righe. A titolo di confronto, il numero di atomi dell'intero universo osservabile è meno di 10^{80} . L'enorme dimensione di queste tabelle significa che (a) nessun agente fisico nell'universo avrà mai lo spazio necessario per memorizzarle, (b) il progettista non avrà mai il tempo di crearle e (c) nessun agente avrà mai il tempo di apprendere le righe corrette in base all'esperienza.

Nonostante tutto ciò, il nostro AGENTE-CON-TABELLA fa quello che vogliamo, assumendo che la tabella sia riempita in modo corretto: implementa effettivamente la funzione agente desiderata.



La sfida principale dell'IA sta nel trovare il modo di scrivere programmi che, nella massima misura possibile, producano un comportamento razionale con una piccola quantità di codice anziché con un'enorme tabella.

Molti casi dimostrano che quest'obiettivo può essere ottenuto in altre discipline: per esempio, le grandi tabelle di radici quadrate usate sia dai bambini che dagli ingegneri prima degli anni 1970 sono state sostituite da un programma di cinque righe che implementa il metodo di Newton nelle calcolatrici tascabili. La domanda è questa: è possibile che l'IA faccia per il comportamento intelligente ciò che Newton ha fatto per le radici quadrate? Noi crediamo di sì.

Nel resto di questo paragrafo delineeremo quattro tipi base di programma agente che contengono i principî alla base di quasi tutti i sistemi intelligenti:

- agenti reattivi semplici;
- agenti reattivi basati su modello;
- agenti basati su obiettivi;
- agenti basati sull'utilità.

Ciascun tipo di programma agente combina componenti particolari in modi particolari per generare azioni. Nel Paragrafo 2.4.6 spiegheremo, in termini generali, come convertire tutti questi agenti in *agenti capaci di apprendere*, che possono migliorare le prestazioni dei propri componenti in modo da generare azioni migliori. Infine, nel Paragrafo 2.4.7 descriveremo la

varietà di modi in cui i componenti stessi possono essere rappresentati all'interno dell'agente; tale varietà fornisce un importante principio di organizzazione per la disciplina e per questo stesso libro.

2.4.2 Agenti reattivi semplici

agente reattivo semplice

Il tipo più semplice è l'**agente reattivo semplice**. Questi agenti scelgono le azioni sulla base della percezione *corrente*, ignorando tutta la storia percettiva precedente. Per esempio, l'aspirapolvere della Figura 2.3 è un agente reattivo semplice, perché la sua decisione è basata unicamente sulla posizione corrente e sul fatto che questa contenga dello sporco o no. La Figura 2.8 mostra uno dei suoi possibili programmi agente.

Noteate che il programma agente dell'aspirapolvere è molto più piccolo della tabella corrispondente. La riduzione più importante deriva dall'aver ignorato la storia delle percezioni, cosa che riduce il numero di sequenze percettive rilevanti da 4^T a solo 4. Un'ulteriore, piccola riduzione deriva dal fatto che, quando il riquadro corrente è sporco, l'azione non dipende dalla posizione. Abbiamo scritto il programma agente utilizzando istruzioni if-then-else, ma è talmente semplice da poter essere implementato anche come circuito booleano.

Comportamenti reattivi semplici si verificano anche in ambienti più complessi. Immaginatevi alla guida del taxi automatizzato. Se la macchina davanti a voi comincia a frenare, e si accendono le relative luci di segnalazione, dovreste notare questo fatto e cominciare anche voi a frenare. In altre parole, devono essere svolti alcuni calcoli sull'input visivo per stabilire la condizione che descriviamo con la frase “la macchina davanti sta frenando”. Questo farà poi scattare, all'interno del programma agente, una connessione con l'azione “inizia a frenare”. Questa connessione prende il nome di **regola condizione–azione**⁶ e si scrive:

if *la-macchina-davanti-sta-frenando* **then** *inizia-a-frenare*.

regola condizione–azione

Anche gli esseri umani fanno uso di connessioni analoghe, alcune delle quali derivano dall'apprendimento (come nel caso della guida), altre invece da riflessi innati (come chiudere le palpebre quando qualcosa si avvicina rapidamente agli occhi). Nel prosieguo del libro vedremo molti modi diversi di apprendere queste connessioni e implementarle in un programma.

Il programma nella Figura 2.8 è scritto *ad hoc*, specificatamente per un ambiente. Un approccio più generale e flessibile consiste nel costruire per prima cosa un interprete di regole condizione–azione di uso generale e poi scrivere gli insiemi di regole specifiche per i vari ambienti operativi. La Figura 2.9 illustra in forma schematica la struttura di questo programma generale, mostrando come le regole condizione–azione permettono all'agente di stabilire

```
function AGENTE-REATTIVO-ASPIRAPOLVERE([posizione, stato]) returns un'azione

  if stato = Sporco then return Aspira
  else if posizione = A then return Destra
  else if posizione = B then return Sinistra
```

Figura 2.8 Il programma agente per l'agente reattivo semplice nell'ambiente dell'aspirapolvere a due stati. Questo programma implementa la funzione agente rappresentata dalla tabella della Figura 2.3.

⁶ Queste regole sono anche chiamate **regole situazione-azione**, **produzioni** o **regole if-then**.

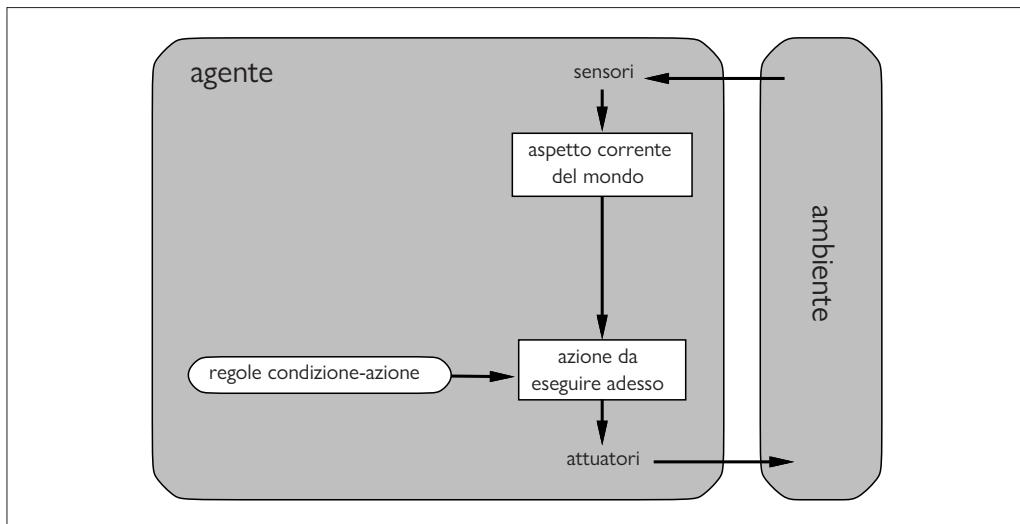


Figura 2.9
Diagramma schematico di un agente reattivo semplice. I rettangoli denotano lo stato interno corrente del processo decisionale dell'agente, mentre gli ovali rappresentano le informazioni di base utilizzate nel processo.

```

function AGENTE-REATTIVO-SEMPLICE(percezione) returns un'azione
persisten: regole, un insieme di regole condizione-azione

stato  $\leftarrow$  INTERPRETA-INPUT(percezione)
regola  $\leftarrow$  REGOLA-CORRISPONDENTE(stato, regole)
azione  $\leftarrow$  regola.AZIONE
return  $\leftarrow$  azione

```

Figura 2.10
Un agente reattivo semplice che agisce secondo una regola la cui condizione corrisponde allo stato corrente, indicato dalla percezione.

una connessione da percezione ad azione (se tutto questo vi sembra banale, non preoccupatevi; ben presto si farà più interessante).

Un programma agente per lo schema della Figura 2.9 è mostrato nella Figura 2.10. La funzione INTERPRETA-INPUT genera una descrizione astratta dello stato corrente partendo dalla percezione, mentre la funzione REGOLA-CORRISPONDENTE restituisce la prima regola dell'insieme che corrisponde a tale descrizione. Notate che la descrizione del programma in termini di “regole” e “corrispondenza” è totalmente astratta; come si è osservato in precedenza, le effettive implementazioni possono essere molto più semplici, come una serie di porte logiche che implementano in hardware un circuito booleano. In alternativa si può usare anche un circuito “neurale” dove le porte logiche sono sostituite dalle unità non lineari di reti neurali artificiali (cfr. il Capitolo 21 del Volume 2).

Gli agenti reattivi semplici hanno l'ammirevole proprietà di essere, appunto, semplici, ma la loro intelligenza è molto limitata. L'agente nella Figura 2.10 funzionerà solo se si può selezionare la decisione corretta in base alla sola percezione corrente, ovvero solo nel caso in cui l'ambiente sia completamente osservabile. Anche una minima parte di non-osservabilità può causare grandi problemi. Per esempio, la regola per frenare che abbiamo formulato precedentemente dà per scontato che la condizione *la-macchina-davanti-sta-frenando* possa essere determinata partendo dalla percezione corrente (un singolo fotogramma). Funziona se la macchina davanti a noi ha una luce di segnalazione della frenata montata centralmente (e quindi identificabile in modo univoco), ma sfortunatamente i modelli più vecchi hanno diverse configurazioni di luci di posizione, frecce e segnalatori di frenata, e non sempre si può stabilire da una singola immagine se la macchina sta frenando o ha semplicemente le luci po-



steriori di posizione accese. Un agente reattivo semplice posto alla guida dietro una macchina siffatta continuerebbe a frenare senza ragione o, ancora peggio, non frenerebbe mai.

Possiamo riscontrare un problema simile anche nel mondo dell’aspirapolvere. Supponiamo che un agente reattivo semplice sia privato del suo sensore di posizione e abbia solo il rilevatore di sporco. Quest’agente avrebbe due sole possibili percezioni: [Sporco] e [Pulito]. In risposta a [Sporco] l’agente può *Aspirare*; ma cosa dovrebbe fare in risposta a [Pulito]? Se per caso è partito nel riquadro A, muoversi a *Sinistra* fallirebbe (per sempre); alla stessa maniera sarebbe errato (per sempre) scegliere di muoversi a *Destra* se l’agente è inizialmente posizionato nel riquadro B. Spesso gli agenti reattivi semplici non sono in grado di evitare cicli infiniti quando operano in ambienti parzialmente osservabili.

randomizzazione

Evitare i cicli infiniti è possibile quando l’agente è in grado di **randomizzare** le sue azioni, scegliendone una in modo casuale. Per esempio, quando l’agente aspirapolvere percepisce [Pulito], potrebbe tirare una moneta per decidere se muoversi a *Destra* o a *Sinistra*. È facile dimostrare che, in media, l’agente raggiungerà l’altro riquadro in due passi. A questo punto, se il riquadro è sporco lo potrà pulire, e la sua attività sarà terminata. Di conseguenza un agente reattivo semplice randomizzato può comportarsi meglio del suo corrispondente deterministico.

Abbiamo menzionato nel Paragrafo 2.3 che, in alcuni sistemi multiagente, un comportamento che include elementi casuali può essere razionale. Negli ambienti ad agente singolo la randomizzazione solitamente *non* è razionale: si può sfruttare come “trucco” per aiutare un agente reattivo semplice a districarsi in certe situazioni, ma nella maggior parte dei casi si può fare molto meglio ricorrendo ad agenti deterministici più sofisticati.

2.4.3 Agenti reattivi basati su modello

stato interno

Il modo più efficace di gestire l’osservabilità parziale, per un agente, è *tener traccia della parte del mondo che non può vedere nell’istante corrente*. Questo significa che l’agente deve mantenere una sorta di **stato interno** che dipende dalla storia delle percezioni e che quindi riflette almeno una parte degli aspetti non osservabili dello stato corrente. Ritornando al problema della frenata, lo stato interno non è troppo complesso: basta mantenere in memoria il fotogramma precedente “acquisito” dalla telecamera, permettendo così all’agente di accorgersi quando due luci rosse alle estremità laterali del veicolo si accendono o spengono contemporaneamente. Per altri compiti, come il cambio di corsia, l’agente deve tenere traccia della posizione di tutte le macchine che non può vedere direttamente. E perché sia possibile guidare, l’agente deve tenere traccia di dove sono le chiavi.

Aggiornare l’informazione dello stato interno al passaggio del tempo richiede che il programma agente possieda due tipi di conoscenza. Prima di tutto, deve avere informazioni sull’evoluzione del mondo nel tempo, suddivisibili approssimativamente in due parti: gli effetti delle azioni dell’agente e le modalità di evoluzione del mondo indipendentemente dall’agente. Per esempio, quando l’agente gira il volante a destra la macchina svolta in quella direzione, e quando piove, la telecamera dell’automobile può bagnarsi. Questa conoscenza sul “funzionamento del mondo”, implementata mediante semplici circuiti logici o sviluppata in una teoria scientifica completa, viene chiamata **modello di transizione** del mondo.

modello di transizione

In secondo luogo, ci servono in formazioni su come lo stato del mondo si rifletta nelle percezioni dell’agente. Per esempio, quando la macchina davanti inizia a frenare, una o più aree rosse illuminate appaiono nell’immagine della telecamera anteriore, e quando la telecamera si bagna, nell’immagine appaiono oggetti a forma di goccia che nascondono parzialmente la strada. Questo tipo di conoscenza è chiamato **modello sensoriale**.

modello sensoriale

Il modello di transizione e il modello sensoriale, insieme, consentono a un agente di tenere traccia dello stato del mondo, per quanto possibile date le limitazioni dei sensori. Un agente che utilizza tali modelli prende il nome di **agente basato su modello**.

agente basato su modello

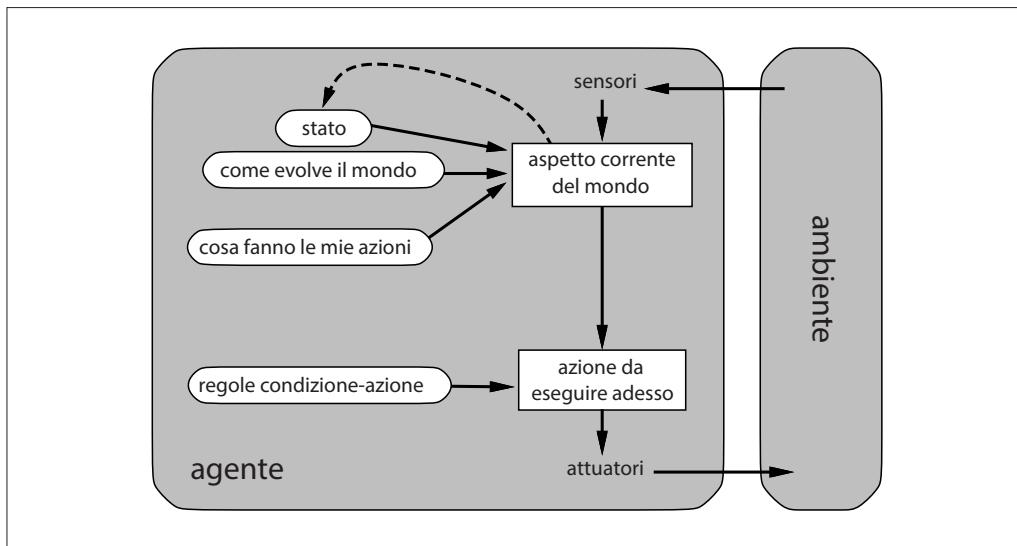


Figura 2.11
Un agente reattivo basato su modello.

```

function AGENTE-REATTIVO-BASATO-SU-MODELLO(percezione) returns un'azione
  persistente: stato, la concezione corrente dello stato del mondo da parte dell'agente
    modello_transizione, una descrizione della dipendenza dello stato successivo dallo
    stato corrente e dall'azione
    modello_sensoriale, una descrizione di come lo stato del mondo attuale è riflesso
    nelle percezioni dell'agente
    regole, un insieme di regole condizione-azione
    azione, l'azione più recente, inizialmente nessuna
  stato  $\leftarrow$  AGGIORNA-STATO(stato, azione, percezione, modello_transizione, modello_sensoriale)
  regola  $\leftarrow$  REGOLA-CORRISPONDENTE(stato, regole)
  azione  $\leftarrow$  regola.AZIONE
  return  $\leftarrow$  azione

```

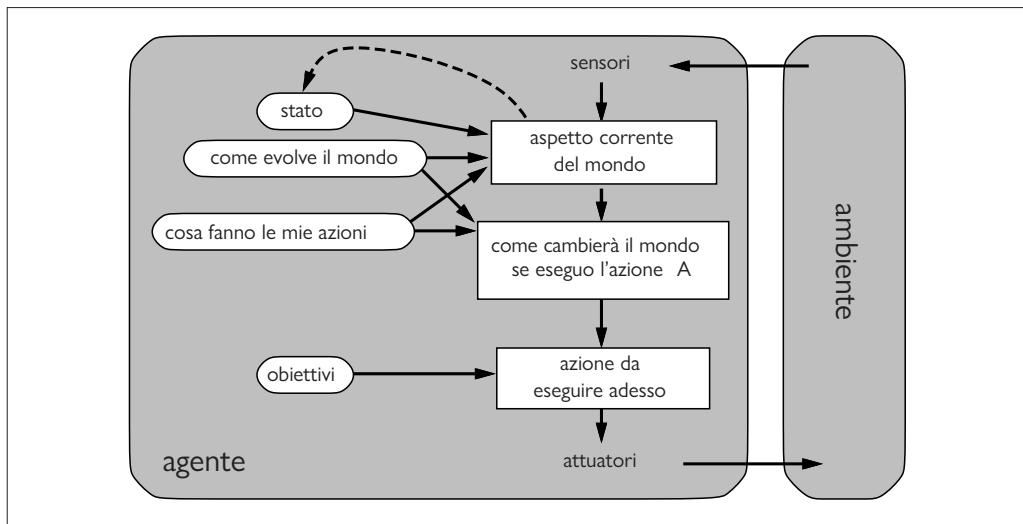
Figura 2.12 Un agente reattivo basato su modello, che tiene traccia dello stato corrente del mondo mediante uno stato interno. A parte questo, l'agente sceglie l'azione da eseguire come un normale agente reattivo.

La Figura 2.11 illustra la struttura di un agente reattivo basato su modello e dotato di stato interno, mostrando come la descrizione aggiornata dello stato scaturisce dalla combinazione del vecchio stato interno e della percezione corrente. Il programma agente è riportato nella Figura 2.12. La parte interessante è la funzione AGGIORNA-STATO, responsabile della creazione del nuovo stato interno. I dettagli della rappresentazione di modelli e stati variano in base al tipo di ambiente e alla particolare tecnologia utilizzata nella progettazione dell'agente.

Indipendentemente dal tipo di rappresentazione utilizzato, talvolta l'agente ha la possibilità di determinare *con esattezza* lo stato corrente di un ambiente parzialmente osservabile. Il rettangolo etichettato “aspetto corrente del mondo” nella Figura 2.11 rappresenta, invece, la “migliore ipotesi” (o talvolta le migliori ipotesi) dell'agente. Per esempio, un taxi automatizzato potrebbe non essere in grado di vedere oltre un enorme camion fermatosi davanti alla vettura, e in questo caso potrebbe soltanto ipotizzare la causa dell'ingorgo. Quindi, l'incertezza riguardo lo stato corrente potrebbe essere inevitabile, ma l'agente deve comunque prendere una decisione.

Figura 2.13

Un agente basato su modello che raggiunge degli obiettivi. Tiene traccia dello stato corrente dell'ambiente e dell'insieme di obiettivi da raggiungere e sceglie l'azione che lo porterà (prima o poi) a soddisfarli.



2.4.4 Agenti basati su obiettivi

obiettivo

ricerca
pianificazione

Conoscere lo stato corrente dell'ambiente non sempre basta a decidere che cosa fare. Per esempio, arrivati a un incrocio il taxi può girare a sinistra, a destra oppure continuare dritto; la decisione corretta dipende dalla sua destinazione. In altre parole, oltre che della descrizione dello stato corrente l'agente ha bisogno di qualche tipo di informazione riguardante il suo **obiettivo** (*goal*), che descriva situazioni desiderabili, come per esempio raggiungere una particolare destinazione. Il programma agente può unire quest'informazione al modello (la stessa informazione usata dagli agenti reattivi basati su modello) per scegliere le azioni che portano al soddisfacimento dell'obiettivo. La Figura 2.13 illustra la struttura di un agente basato su obiettivi.

Talvolta scegliere un'azione in base a un obiettivo è molto semplice, quando questo può essere raggiunto in un solo passo. Altre volte è più difficile, per esempio quando l'agente deve considerare lunghe sequenze di azioni per trovare il “cammino” che porta al risultato agognato. La **ricerca** (Capitoli dal 3 al 5) e la **pianificazione** (Capitolo 11) sono sottocampi dell'IA dedicati proprio a identificare le sequenze di azioni che permettono a un agente di raggiungere i propri obiettivi.

Notate che questo tipo di decisioni non ha nulla a che vedere con le regole condizione-azione che abbiamo descritto prima, perché ora dobbiamo prendere in considerazione il futuro sotto due aspetti: “cosa accadrà se faccio così e cosà?” e anche “se faccio questo sarò soddisfatto?” Nella progettazione degli agenti reattivi, questa informazione non viene rappresentata esplicitamente, perché le regole fornite internamente mettono direttamente in corrispondenza percezioni e azioni. L'agente reattivo frena quando vede le luci di frenata, senza rendersi conto del motivo. Un agente basato su obiettivi frena quando vede le luci di frenata perché quella è l'unica azione che, secondo la sua previsione, permette di raggiungere l'obiettivo di non tamponare altre macchine.

Benché un agente basato su obiettivi sembri meno efficiente, d'altra parte è più flessibile, perché la conoscenza che guida le sue decisioni è rappresentata esplicitamente e può essere modificata. Per esempio, il comportamento dell'agente basato su obiettivi può essere facilmente alterato per farlo andare verso una destinazione diversa, semplicemente specificando tale destinazione come obiettivo. Le regole dell'agente reattivo che governano quando sterzare e quando andare dritto, invece, funzioneranno solo per una singola destinazione finale; per andare da qualche altra parte dovranno essere tutte riscritte.

2.4.5 Agenti basati sull'utilità

Nella maggior parte degli ambienti gli obiettivi, da soli, non bastano a generare un comportamento di alta qualità. Per esempio, ci sono molte sequenze di azioni che porteranno un taxi alla sua destinazione (soddisfacendo così il suo obiettivo) ma alcune sono più veloci, sicure, affidabili o economiche di altre. Gli obiettivi forniscono solamente una distinzione binaria tra stati “contenti” e “scontenti”, laddove una misura di prestazione più generale dovrebbe permettere di confrontare stati del mondo differenti e misurare precisamente la contentezza che potrebbero portare all'agente. Dato che “contentezza” non suona molto scientifico, economisti e informatici utilizzano il termine **utilità**.

Abbiamo già visto che una misura di prestazione assegna un punteggio a qualsiasi sequenza di stati dell'ambiente, così da poter facilmente distinguere tra modi più o meno desiderabili di raggiungere la destinazione del taxi. Una **funzione di utilità** di un agente è, in sostanza, un'internalizzazione della misura di prestazione. Purché la funzione di utilità interna e la misura di prestazione esterna concordino, un agente che sceglie le azioni per massimizzare l'utilità sarà razionale in base alla misura di prestazione esterna.

Sottolineiamo ancora che questo non è il *solo* modo di essere razionali (abbiamo già visto un programma agente razionale per il mondo dell'aspirapolvere, descritto nella Figura 2.8, che non ha idea alcuna della propria funzione di utilità), ma, come gli agenti basati su obiettivi, un agente basato sull'utilità presenta molti vantaggi in termini di flessibilità e apprendimento. Inoltre, in due categorie di casi, gli obiettivi sono inadeguati ma un agente basato sull'utilità è in grado di prendere decisioni razionali. Prima di tutto, quando ci sono più obiettivi in conflitto che non si possono soddisfare tutti insieme (per esempio, velocità e sicurezza), la funzione di utilità specifica come bilanciarli. In secondo luogo, quando ci sono più obiettivi raggiungibili ma nessuno può essere ottenuto con certezza, il concetto di utilità fornisce un mezzo per confrontare le probabilità di successo e l'importanza degli obiettivi.

Situazioni di parziale osservabilità e non determinismo sono molto diffuse nel mondo reale, e di conseguenza anche i casi in cui occorre prendere decisioni in condizioni di incertezza sono comuni. In termini tecnici, un agente razionale basato sull'utilità sceglie l'azione che massimizza l'**utilità attesa** dei risultati, ovvero l'utilità che l'agente si attende di ottenere, in media, date le probabilità e le utilità di ciascun risultato (nell'Appendice A viene definito in maniera più precisa il concetto di valore atteso). Nel Volume 2 mostreremo che ogni agente razionale deve comportarsi *come se* possedesse una funzione di utilità di cui cerca di massimizzare il valore atteso. Un agente che possiede una funzione di utilità *esplicita* può prendere decisioni razionali, seguendo un algoritmo generale che non dipende dalla specifica funzione di utilità che si desidera massimizzare. In questo modo la definizione “globale” di razionalità, che definisce razionali quelle funzioni agente che hanno le prestazioni migliori, viene trasformata in un vincolo “locale” in progetti di agenti razionali che possono essere espressi in un semplice programma.

La Figura 2.14 mostra la struttura di un agente basato sull'utilità. I programmi agente basati sull'utilità saranno trattati nei Capitoli 16 e 17, in cui progetteremo agenti capaci di prendere decisioni che considerano l'incertezza intrinseca in ambienti non deterministici o parzialmente osservabili. I processi decisionali in ambienti multiagente sono anch'essi studiati nel quadro della teoria dell'utilità, come verrà spiegato nel Capitolo 18.

A questo punto il lettore potrebbe chiedersi: “È tutto così semplice? È sufficiente costruire agenti che massimizzino l'utilità attesa?”. È vero che tali agenti sarebbero intelligenti, ma non è tutto così semplice. Un agente basato sull'utilità deve modellare e tenere traccia del proprio ambiente, e questi compiti hanno comportato lunghe e approfondite ricerche su percezione, rappresentazione, ragionamento e apprendimento. I risultati di queste ricerche occupano molti capitoli di questo libro. Scegliere il corso d'azione che massimizza l'utilità è anch'esso un compito difficile che richiede algoritmi ingegnosi, i quali occupano altri capitoli

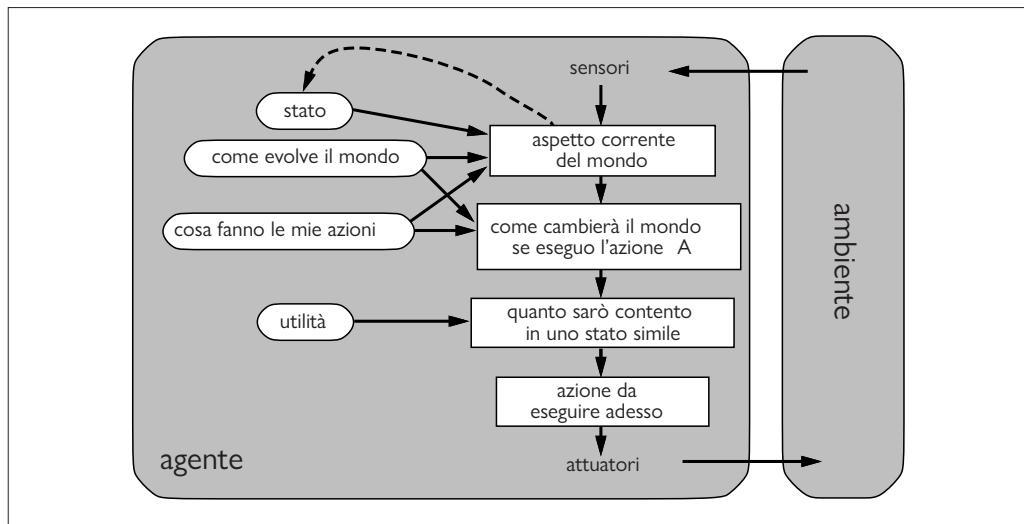
utilità

funzione di utilità

utilità attesa

Figura 2.14

Un agente basato su modello che massimizza l'utilità. Oltre a tener traccia dello stato dell'ambiente, l'agente impiega una funzione utilità che misura le sue preferenze tra i vari stati del mondo. L'azione prescelta è quella che massimizza l'utilità attesa, calcolata come media dei valori di utilità degli stati possibili pesati con la rispettiva probabilità di verificarsi.



di questo libro. Anche con questi algoritmi, comunque, in genere la razionalità perfetta è irraggiungibile nella pratica, a causa della complessità computazionale, come abbiamo sottolineato nel Capitolo 1. Osserviamo inoltre che non tutti gli agenti basati sull'utilità sono basati su modello; nei Capitoli 22 e 26 del Volume 2 vedremo che un **agente model-free** (privo di modello) può apprendere qual è l'azione migliore in una particolare situazione senza mai apprendere esattamente in quale modo tale azione modifichi l'ambiente.

Infine, tutto ciò si basa sull'assunto che il progettista sia in grado di specificare correttamente la funzione di utilità; nei Capitoli 17, 18 del presente volume e 22 del Volume 2 esamineremo in modo più approfondito il problema delle funzioni di utilità ignote.

2.4.6 Agenti capaci di apprendere

Fin qui abbiamo descritto programmi agente che usano vari metodi per scegliere le azioni. Non abbiamo ancora spiegato in che modo *nascono* i programmi agente. Nel suo famoso articolo, Turing (1950) considera l'idea di programmare a mano le sue macchine intelligenti. Dopo aver stimato il lavoro necessario, conclude: “sembra auspicabile un metodo più rapido”. Il metodo proposto dallo stesso Turing è costruire macchine capaci di apprendere e poi addestrarle. In molti campi dell'IA, questo è oggi il metodo preferito per creare sistemi allo stato dell'arte. Qualsiasi tipo di agente (basato su modello, su obiettivi, su utilità e così via) può essere costruito come agente capace di apprendere (o meno).

L'apprendimento presenta un altro vantaggio: permette agli agenti di operare in ambienti inizialmente sconosciuti, diventando col tempo più competenti di quanto fossero all'inizio, allorché si basavano sulla sola conoscenza iniziale. In questo paragrafo presenteremo brevemente i concetti principali dell'apprendimento. In tutto il libro sono presenti commenti sulla possibilità e i metodi per l'apprendimento in particolari categorie di agenti. Nei Capitoli 19–22 del Volume 2 analizzeremo in modo approfondito gli algoritmi di apprendimento.

Un agente capace di apprendere può essere diviso in quattro componenti astratti, come si vede nella Figura 2.15. La distinzione più importante è tra l'**elemento di apprendimento** (*learning element*), che è responsabile del miglioramento interno, e l'**elemento esecutivo** (*performance element*), che si occupa della selezione delle azioni esterne. Quest'ultimo è ciò che abbiamo considerato fin qui come se costituisse l'intero agente: prende in input le percezioni e decide le azioni. L'elemento di apprendimento utilizza informazioni provenienti

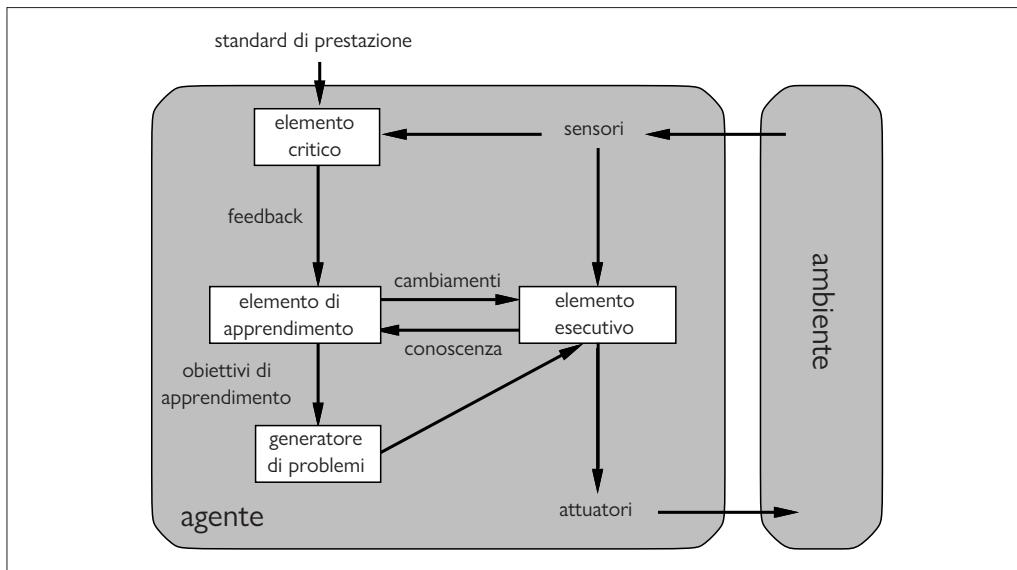


Figura 2.15
Un modello generale di agente capace di apprendere. Il riquadro “elemento esecutivo” rappresenta ciò che in precedenza abbiamo considerato come l’intero programma agente. Ora, il riquadro “elemento di apprendimento” può modificare il programma per migliorarne la prestazione.

dall'**elemento critico** riguardo le prestazioni correnti dell’agente e determina se e come modificare l’elemento esecutivo affinché in futuro si comporti meglio.

Il progetto dell’elemento di apprendimento dipende molto da quello dell’elemento esecutivo. Quando si cerca di progettare un agente che impara a svolgere una certa attività, la prima domanda da porsi non è “come faccio a fargli apprendere questa cosa?” ma “quale tipo di elemento esecutivo permetterà al mio agente di fare questa cosa, una volta che l’avrà appresa?”. Dato un progetto di elemento esecutivo, si possono costruire meccanismi di apprendimento per migliorare ogni parte dell’agente.

L’elemento critico dice a quello di apprendimento come si sta comportando l’agente rispetto a uno standard di prestazione prefissato. Quest’elemento è necessario perché le percezioni, in sé, non forniscono alcuna indicazione del successo dell’agente. Un programma di scacchi potrebbe ricevere una percezione che indica che ha dato scacco matto all’avversario, ma ha bisogno di uno standard di prestazione per sapere che questa è una cosa buona; la percezione da sola non basta. È importante che lo standard sia prefissato: concettualmente lo si può pensare come un’entità del tutto separata dall’agente, dato che quest’ultimo non lo deve modificare per adattarlo al suo comportamento.

L’ultimo componente di un agente capace di apprendere è il **generatore di problemi**, il cui scopo è suggerire azioni che portino a esperienze nuove e significative. Se si lasciasse mano libera all’elemento esecutivo, esso continuerebbe a ripetere le azioni che ritiene migliori date le conoscenze attuali. Ma se l’agente è disposto a esplorare qualche altra possibilità, e magari eseguire nel breve termine qualche azione subottima, potrebbe scoprire l’esistenza di azioni migliori a lungo termine. Scopo del generatore di problemi è suggerire tali azioni esplorative. Questo è ciò che fanno gli scienziati quando svolgono esperimenti: Galileo non pensava che far cadere sassi dalla cima di una torre, a Pisa, fosse un’azione valida in sé. Non stava cercando di rompere i sassi, né di modificare il cervello di qualche sfortunato passante. Lo scopo era modificare il suo stesso cervello, formulando una teoria migliore sul moto degli oggetti.

L’elemento di apprendimento può modificare uno qualsiasi dei componenti “di conoscenza” mostrati nei diagrammi degli agenti (Figure 2.9, 2.11, 2.13 e 2.14). Nei casi più semplici l’apprendimento scaturisce direttamente dalla sequenza percettiva. L’osservazione di coppie di stati successivi dell’ambiente permette all’agente di imparare “cosa fanno le mie azioni” e

elemento critico

generatore di problemi

ricompensa
penalità

**rappresentazione
atomica**

“come evolve il mondo” a seguito delle sue azioni. Per esempio, se il taxi automatizzato esercita una pressione eccessiva sui freni mentre sta guidando su una strada bagnata, imparerà ben presto quanta decelerazione si ottiene in realtà e se le ruote slittano. Il generatore di problemi potrebbe individuare alcune parti del modello che devono essere migliorate e suggerire degli esperimenti, quali provare a frenare su diversi tipi di strade e in condizioni ambientali differenti.

Migliorare i componenti di un agente basato su modello in modo che siano più aderenti alla realtà è quasi sempre utile, indipendentemente dallo standard di prestazione esterno (in alcuni casi è meglio, da un punto di vista computazionale, avere un modello semplice ma leggermente impreciso piuttosto che un modello perfetto ma eccessivamente complesso). Quando si cerca di apprendere un componente reattivo o una funzione di utilità è necessario disporre di informazioni fornite dallo standard esterno.

Per esempio, supponiamo che l’agente tassista non riceva mai alcuna mancia dai passeggeri che sono stati rozzamente sballottati per tutto il viaggio. Lo standard di prestazione esterno deve informare l’agente che la perdita di tutte le mance rappresenta un contributo negativo alla sua performance globale; l’agente potrebbe allora apprendere che la guida nervosa non contribuisce alla sua utilità. In un certo senso, lo standard di prestazione caratterizza una parte delle percezioni come **ricompense** (o rispettivamente **penalità**) che rappresentano un feedback diretto sulla qualità del comportamento dell’agente. In questo modo si possono descrivere standard di prestazione innati, quali il dolore e la fame negli animali.

In generale, le *scelte umane* possono fornire informazioni sulle preferenze umane. Per esempio, supponiamo che il taxi non sappia che le persone generalmente non amano i rumori forti, e si fissi sull’idea di suonare di continuo il clacson per assicurarsi che i pedoni siano avvertiti del suo arrivo. Il comportamento umano conseguente – coprirsi le orecchie, dire parolacce e magari tagliare i fili del clacson – fornirebbe all’agente evidenze con cui aggiornare la sua funzione di utilità. Questo aspetto è approfondito nel Capitolo 22 del Volume 2.

Riassumendo, gli agenti sono formati da una varietà di componenti che possono essere rappresentati in molti modi nel programma agente. Questo fa sì che anche nei metodi di apprendimento sia presente una grande diversità. C’è comunque un tema unificante: l’apprendimento in un agente intelligente può essere definito come il processo che modifica ogni suo componente affinché si accordi meglio con l’informazione di feedback disponibile, migliorando così le prestazioni globali dell’agente.

2.4.7 Funzionamento dei componenti dei programmi agente

Secondo la nostra descrizione (di livello molto alto) i programmi agente sono costituiti da vari componenti, la cui funzione è quella di rispondere a domande come: “Qual è l’aspetto corrente del mondo?”, “Qual è l’azione da eseguire adesso?”, “Che cosa fanno le mie azioni?”. Per chi studia l’IA la prossima domanda è: “Come funzionano questi componenti in pratica?”. Servirebbero mille pagine soltanto per iniziare a rispondere a questa domanda, ma qui vogliamo portare l’attenzione del lettore su alcune distinzioni di base tra i vari modi in cui i componenti possono rappresentare l’ambiente abitato dall’agente.

In termini un po’ approssimativi, possiamo disporre le rappresentazioni atomica, fattorizzata e strutturata lungo un asse di complessità e potenza espressiva crescente. Per illustrare questi concetti è utile considerare un particolare componente, come quello che affronta la domanda: “Che cosa fanno le mie azioni?”. Questo componente descrive i cambiamenti che potrebbero verificarsi nell’ambiente come conseguenza di un’azione; la Figura 2.16 illustra schematicamente come potrebbero essere rappresentate queste transizioni.

In una **rappresentazione atomica** ogni stato del mondo è indivisibile, non ha struttura interna. Consideriamo il problema di trovare un percorso per viaggiare in auto da un’estremità

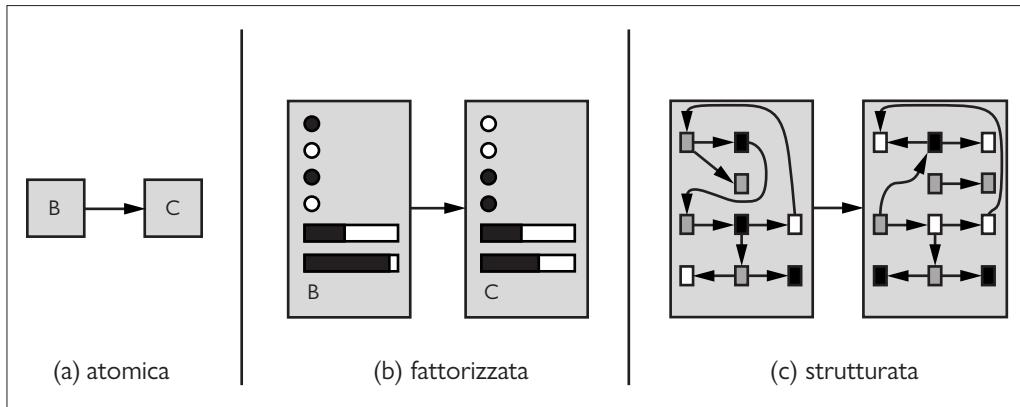


Figura 2.16 Tre modi per rappresentare stati e transizioni fra di loro. (a) Rappresentazione atomica: uno stato (come B o C) è una scatola nera priva di struttura interna. (b) Rappresentazione fattorizzata: uno stato è costituito da un vettore di valori di attributi; tali valori possono essere booleani, numeri reali o un simbolo appartenente a un insieme fissato. (c) Rappresentazione strutturata: uno stato include oggetti, ognuno dei quali può avere attributi propri oltre a relazioni con altri oggetti.

all’altra del paese attraversando una certa sequenza di città (Figura 3.1 del Capitolo 3). Per risolvere questo problema potrebbe essere sufficiente ridurre lo stato dal mondo al nome della città in cui ci troviamo (un singolo atomo di conoscenza); una “scatola nera” la cui sola proprietà discernibile è il fatto di essere uguale o diversa da un’altra scatola nera. Gli algoritmi standard alla base di ricerche e giochi (Capitoli 3–5), modelli di Markov nascosti (Capitolo 14) e processi decisionali di Markov (Capitolo 17) lavorano tutti con rappresentazioni atomiche.

Una **rappresentazione fatorizzata** suddivide ogni stato in un insieme fissato di **variabili attributi**, ognuno dei quali può avere un **valore**. Consideriamo una descrizione più fedele dello stesso problema di guida, in cui dobbiamo preoccuparci non solo della posizione atomica in una città, ma anche di quanto carburante c'è nel serbatoio dell'auto, delle attuali coordinate GPS, del fatto che sia accesa o meno la spia dell'olio, di quanti soldi abbiamo per i pedaggi autostradali, di quale stazione è sintonizzata alla radio e così via. Mentre due stati atomici diversi non hanno nulla in comune (sono semplicemente due scatole nere diverse), due stati fatorizzati diversi possono condividere alcuni attributi (per esempio il fatto di trovarsi in una particolare posizione individuata da coordinate GPS) e non altri (per esempio il fatto di disporre di molto carburante o di averlo esaurito); in questo modo è molto più facile determinare come passare da uno stato a un altro. Molti campi importanti dell'IA si basano su rappresentazioni fatorizzate, tra cui gli algoritmi di soddisfacimento di vincoli (Capitolo 6), calcolo proposizionale (Capitolo 7), pianificazione (Capitolo 11), reti bayesiane (Capitoli 12–16) e vari algoritmi di apprendimento automatico.

In molti casi è necessario interpretare il mondo come se fosse composto da cose che sono in *relazione* tra loro, e non semplicemente da variabili con valori. Per esempio, potremmo notare che un grande camion davanti a noi sta entrando in retromarcia nel vialetto di un'azienda di allevamento, ma c'è una mucca che è scappata e che gli blocca la strada. Difficilmente una rappresentazione fattorizzata disporrà dell'attributo *CamionDavantiInRetromarciaInViale-AziendaAllevamentoBloccatoDaMuccaScappata* con valore *vero* o *falso*. In questi casi ci serve una **rappresentazione strutturata**, in cui è possibile descrivere in modo esplicito oggetti come mucche e camion insieme alle loro relazioni (Figura 2.16c). Le rappresentazioni strutturate sono alla base dei database relazionali e della logica del primo ordine (Capitoli 8, 9 e 10), dei modelli probabilistici del primo ordine (Capitolo 15) e di gran parte dei metodi di

**rappresentazione
fattorizzata
variabili
attributi
valore**

**rappresentazione
strutturata**

espressività

comprensione del linguaggio naturale (Capitoli 23–24 del Volume 2). In effetti, gran parte di ciò che gli uomini esprimono in linguaggio naturale riguarda oggetti e relazioni.

Come abbiamo detto in precedenza, le rappresentazioni atomiche, fattorizzate e strutturate sono disposte lungo un asse di **espressività** crescente. In parole povere, una rappresentazione più espressiva può catturare, con livello di concisione almeno uguale, tutto ciò che è in grado di catturare una rappresentazione meno espressiva. Spesso il linguaggio più espressivo è molto più conciso; per esempio, le regole degli scacchi si possono scrivere in una o due pagine con un linguaggio per la rappresentazione strutturata quale la logica del primo ordine, ma richiedono migliaia di pagine con un linguaggio a rappresentazione fattorizzata quale la logica proposizionale e circa 10^{38} pagine con un linguaggio atomico come quello degli automi a stati finiti. D'altra parte, ragionamento e apprendimento divengono più complessi al crescere della potenza espressiva della rappresentazione. Per ottenere i vantaggi delle rappresentazioni expressive evitando gli svantaggi, i sistemi intelligenti del mondo reale potrebbero avere la necessità di operare su tutti i punti dell'asse contemporaneamente.

rappresentazione locale**rappresentazione distribuita**

Un altro asse di rappresentazione riguarda la corrispondenza di concetti a locazioni di memoria fisica, in un computer o in un cervello. Se esiste una corrispondenza uno a uno tra concetti e locazioni di memoria, parliamo di **rappresentazione locale**. Al contrario, se la rappresentazione di un concetto è sparsa su molte locazioni di memoria, ciascuna delle quali è utilizzata come parte di una rappresentazione di più concetti diversi, parliamo di **rappresentazione distribuita**. Le rappresentazioni distribuite sono più robuste rispetto al rumore e alla perdita di informazioni. Nel caso di una rappresentazione locale, la corrispondenza da concetto a locazione di memoria è arbitraria, e se un errore di trasmissione altera qualche bit, potrebbe confondere il concetto *Traccia* con il concetto *Treccia*, che non c'entra nulla. Con una rappresentazione distribuita, invece, possiamo pensare che ogni concetto rappresenti un punto in uno spazio multidimensionale, e se si alterano pochi bit si passa a un punto vicino che avrà un significato simile.

2.5 Riepilogo

In questo capitolo abbiamo presentato una sorta di panoramica estremamente succinta dell'IA, che abbiamo definito come la scienza della progettazione di agenti. I punti più importanti da ricordare sono i seguenti.

- Un **agente** è qualcosa che percepisce e agisce all'interno di un ambiente. La sua **funzione agente** specifica l'azione intrapresa in risposta a qualsiasi sequenza di percezioni.
- La **misura di prestazione** valuta il comportamento dell'agente in un ambiente. Un **agente razionale** agisce in modo da massimizzare il valore atteso della misura di prestazione, data la sequenza percettiva fino a quel momento.
- La specifica di un **ambiente operativo**, include la misura di prestazione, l'ambiente esterno, gli attuatori e i sensori. Nella progettazione di un agente il primo passo deve sempre consistere nella specifica più dettagliata possibile dell'ambiente operativo.
- Gli ambienti operativi possono essere classificati in base a molte proprietà significative. Possono essere completamente o parzialmente osservabili, a singolo agente o multiagente, deterministici o non deterministici, episodici o sequenziali, statici o dinamici, discreti o continui, noti o ignoti.
- Nei casi in cui la misura di prestazione è ignota o difficile da specificare correttamente, vi è un rischio significativo che l'agente ottimizzi l'obiettivo sbagliato. In tali casi, il progetto dell'agente dovrebbe riflettere l'incertezza relativa all'obiettivo reale.

- Il **programma agente** implementa la funzione agente. Esistono diversi schemi base per i programmi agente, che riflettono il tipo di informazione rappresentata esplicitamente e utilizzata nel processo decisionale. Gli schemi variano in efficienza, compattezza e flessibilità; quello più appropriato per un dato programma agente dipende dalla natura dell'ambiente.
- Gli **agenti reattivi semplici** rispondono direttamente alle percezioni, mentre gli **agenti reattivi basati su modello** mantengono uno stato interno per tener traccia degli aspetti del mondo che non sono visibili nelle percezioni correnti. Gli **agenti basati su obiettivi** agiscono per raggiungere tali obiettivi, e gli **agenti basati sull'utilità** cercano di massimizzare la “contentezza” attesa.
- Tutti gli agenti possono migliorare le loro prestazioni mediante l'**apprendimento**.

Note storiche e bibliografiche

Il ruolo centrale dell’azione nell’intelligenza, ovvero il concetto di ragionamento pratico, risale almeno all’*Etica Nicomachea* di Aristotele. Il ragionamento pratico fu anche l’argomento dell’importante articolo di McCarthy *Programs with common sense* (1958). La robotica e la teoria del controllo, per loro stessa natura, si occupano principalmente di agenti fisici. Il concetto di **controllore** della teoria del controllo è identico a quello di agente nell’IA. In modo forse sorprendente, per la maggior parte della sua storia l’IA si è concentrata su componenti isolati (sistemi di domanda/risposta, dimostratori di teoremi, sistemi di visione e così via) piuttosto che sugli agenti nella loro interezza. Un’importante eccezione fu rappresentata dalla discussione sugli agenti nel libro di Genesereth e Nilsson (1987). La visione basata su agenti oggi è ampiamente accettata e costituisce un tema centrale dei libri più recenti (Padgham e Winikoff, 2004; Jones, 2007; Poole e Mackworth, 2017).

Nel Capitolo 1 abbiamo ritrovato le radici del concetto di razionalità nella filosofia e nell’economia. Nell’IA il concetto rivestì un interesse marginale fino alla metà degli anni 1980, quando cominciò a diffondersi in molte discussioni riguardanti le basi tecniche della disciplina. In un articolo, Jon Doyle (1983) predisse che la progettazione di agenti razionali sarebbe arrivata a essere considerata la missione principale dell’IA, mentre altri argomenti popolari se ne sarebbero separati per formare nuove discipline.

L’attenzione verso le proprietà dell’ambiente e le loro conseguenze sulla progettazione di agenti razionali appare con più evidenza nella tradizione della teoria del controllo: per esempio, i sistemi di controllo classici (Dorf e Bishop, 2004; Kirk, 2004) considerano

ambienti completamente osservabili e deterministici; il controllo ottimo stocastico (Kumar e Varaiya, 1986; Bertsekas e Shreve, 2007) si occupa di ambienti parzialmente osservabili e stocastici; il controllo ibrido (Henzinger e Sastry, 1998; Cassandras e Lygeros, 2006) è rivolto agli ambienti che contengono sia elementi discreti che continui. La distinzione tra ambienti parzialmente e completamente osservabili è centrale nella letteratura che riguarda la **programmazione dinamica**, sviluppata in seno alla ricerca operativa (Puterman, 1994). Ne discuteremo nel Capitolo 17.

Anche se gli agenti reattivi semplici sono stati il modello centrale per gli psicologi comportamentisti (cfr. il Capitolo 1), la maggior parte degli studiosi di IA li considera troppo semplici per fornire reali vantaggi. [Rosenschein (1985) e Brooks (1986) hanno messo in discussione questa ipotesi; cfr. il Capitolo 26]. Notevole impegno è stato profuso per trovare algoritmi efficienti per tenere traccia di ambienti complessi (Bar-Shalom *et al.*, 2001; Choset *et al.*, 2005; Simon, 2006), soprattutto in condizioni probabilistiche.

Gli agenti basati su obiettivi sono già presenti nel concetto aristotelico di ragionamento pratico e attraversano la letteratura fino ai primi articoli di McCarthy sull’IA logica. Il robot Shakey (Fikes e Nilsson, 1971; Nilsson, 1984) fu la prima realizzazione robotica di un agente logico basato su obiettivi. Un’analisi logica degli agenti basati su obiettivi apparì poi in Genesereth e Nilsson (1987), e una metodologia di programmazione basata sugli obiettivi, chiamata *agent-oriented programming*, fu sviluppata da Shoham (1993). L’approccio basato su agenti oggi è estremamente popolare nell’ingegneria del software (Ciancarini e Wooldridge, 2001) e si è diffuso anche nel campo dei sistemi opera-

tivi, dove l'**autonomic computing** fa riferimento a sistemi e reti informatiche in grado di monitorare e controllare loro stessi con un ciclo percezione-azione e metodi di apprendimento automatico (Kephart e Chess, 2003). Poiché si osserva che un insieme di programmi agente progettati per lavorare bene insieme all'interno di un ambiente realmente multiagente ha necessariamente aspetti di modularità (i programmi non condividono lo stato interno e comunicano tra loro soltanto attraverso l'ambiente), nel campo dei **sistemi multiagente** è comune progettare il programma agente di un singolo agente come un insieme di subagenti autonomi. In alcuni casi è anche possibile dimostrare che il sistema risultante produce le stesse soluzioni ottime di un progetto monolitico.

L'approccio basato su obiettivi domina anche la tradizione della psicologia cognitiva nell'area della risoluzione di problemi, a partire da *Human Problem Solving* (Newell e Simon, 1972), che ebbe un'enorme influenza, attraverso tutti i lavori più recenti di Newell (Newell, 1990). Gli obiettivi, ulteriormente suddivisi in *desideri* (generali) e *intenzioni* (perseguite in un dato momento) sono l'aspetto centrale dell'importante teoria degli agenti sviluppata da Michael Bratman (1987).

Come si è osservato nel Capitolo 1, lo sviluppo della teoria dell'utilità come fondamento per il comportamento razionale risale a secoli fa. Nell'IA, le prime ricerche hanno puntato maggiormente sugli obiettivi, con qualche eccezione (Feldman e Sproull, 1977). La rinascita dell'interesse nei metodi probabilistici avvenuta negli anni 1980 ha portato ad accettare la massimizzazione dell'utilità attesa come approccio più generale per il processo decisionale (Horvitz *et al.*, 1988). Il testo di Pearl (1988) è stato il primo a considerare approfonditamente la probabilità e la teoria dell'utilità; la sua esposizione dei metodi pratici per ragionare e prendere decisioni in condizione di incertezza fu probabilmente il singolo fattore più importante nel rapido spostamento dell'attenzione, negli

anni 1990, verso gli agenti basati sull'utilità (cfr. Capitolo 16). La formalizzazione dell'apprendimento con rinforzo nel contesto della teoria delle decisioni ha anch'essa contribuito a questo spostamento (Sutton, 1988). È notevole il fatto che, fino a tempi molto recenti, quasi tutte le ricerche in IA hanno assunto che la misura di prestazione potesse essere specificata in modo esatto e corretto sotto forma di funzione di utilità o funzione di ricompensa (Hadfield-Menell *et al.*, 2017; Russell, 2019).

Lo schema generale per gli agenti capaci di apprendere rappresentato nella Figura 2.15 è un classico della letteratura sull'apprendimento automatico (Buchanan *et al.*, 1978; Mitchell, 1997). Esempi di questo schema, implementati sotto forma di software, risalgono almeno al programma di Arthur Samuel (1959, 1967) per giocare a dama. Gli agenti capaci di apprendere sono discussi nei Capitoli 19–23 del Volume 2.

I primi articoli su approcci basati sugli agenti sono stati raccolti da Huhns e Singh (1998) e Wooldridge e Rao (1999). I testi dedicati ai sistemi multiagente solitamente forniscono una introduzione a molti aspetti della progettazione di agenti (Weiss, 2000; Wooldridge, 2002). A partire dagli anni 1990 si sono tenute varie serie di congressi dedicati agli agenti, tra cui l'International Workshop on Agent Theories, Architectures, and Languages (ATAL), l'International Conference on Autonomous Agents (AGENTS) e l'International Conference on Multi-Agent Systems (ICMAS). Nel 2002 questi tre congressi si sono fusi per formare l'International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). Dal 2000 al 2012 si sono tenuti workshop annuali sull'Agent-Oriented Software Engineering (AOSE). La rivista *Autonomous Agents and Multi-Agent Systems* è stata fondata nel 1998. Infine, *Dung Beetle Ecology* (Hanski e Cambefort, 1991) fornisce una quantità di informazioni interessanti sul comportamento degli scarabei stercorei, per cui si trovano diversi video illustrativi su YouTube.

P A R T E

2

Risoluzione di problemi

**Capitolo 3 Risolvere i problemi
con la ricerca**

Capitolo 4 Ricerca in ambienti complessi

Capitolo 5 Ricerca con avversari e giochi

**Capitolo 6 Problemi di soddisfacimento
di vincoli**



CAPITOLO

3

- 3.1 Agenti risolutori di problemi
- 3.2 Problemi esemplificativi
- 3.3 Algoritmi di ricerca
- 3.4 Strategie di ricerca non informata
- 3.5 Strategie di ricerca informata o euristica
- 3.6 Funzioni euristiche
- 3.7 Riepilogo
- Note storiche e bibliografiche

Risolvere i problemi con la ricerca

In cui vediamo come un agente può guardare in avanti per trovare una sequenza di azioni che alla fine raggiungerà i suoi scopi.

Quando l'azione giusta da compiere non è subito evidente, un agente può avere la necessità di *guardare in avanti*, cioè considerare una *sequenza* di azioni che formano un cammino che porterà a uno stato obiettivo. Questo tipo di agente è chiamato **agente risolutore di problemi** e il processo computazionale che effettua è una **ricerca**.

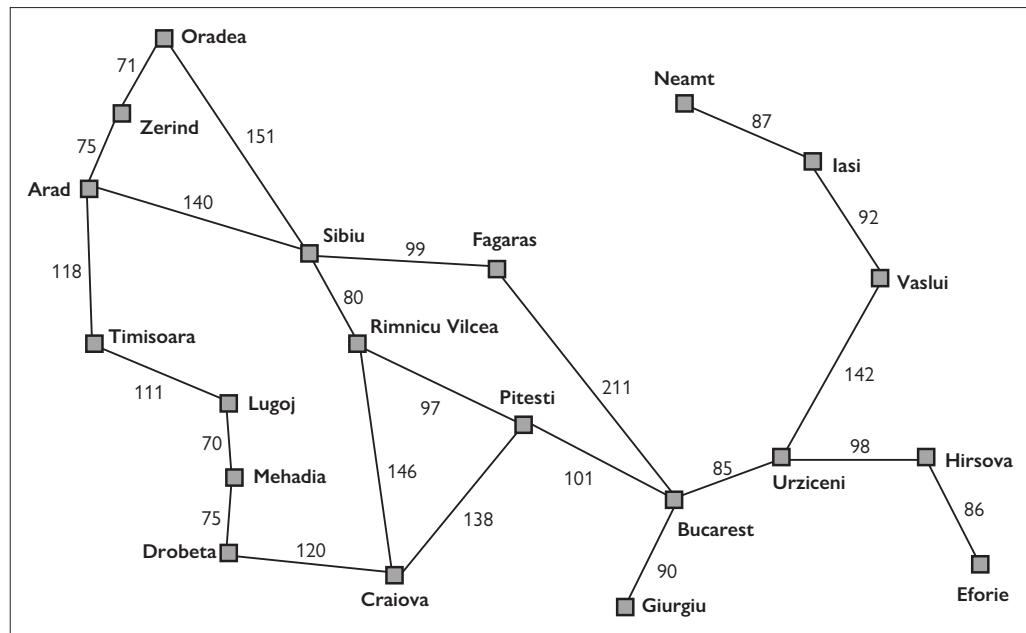
Gli agenti risolutori di problemi utilizzano rappresentazioni **atomiche** (cfr. Paragrafo 2.4.7) in cui gli stati del mondo sono considerati come entità prive di una struttura interna visibile agli algoritmi per la risoluzione dei problemi. Gli agenti che utilizzano rappresentazioni di stati **fattorizzate** o **struturate** sono solitamente chiamati **agenti pianificatori** e sono discussi nei Capitoli 7 e 11.

Esamineremo diversi algoritmi di ricerca. In questo capitolo consideriamo soltanto gli ambienti più semplici: episodici, a singolo agente, completamente osservabili, deterministici, statici, discreti e noti. Distinguendo tra algoritmi **informati**, in cui l'agente è in grado di stimare la distanza dall'obiettivo, e **non informati**, in cui non c'è la disponibilità di tale stima. Nel Capitolo 4 esamineremo ambienti con meno vincoli e nel Capitolo 5 considereremo ambienti con agenti multipli.

Questo capitolo utilizza il concetto di complessità asintotica – notazione $O(n)$. Chi non ha familiarità con questo concetto dovrebbe consultare l'Appendice A.

Figura 3.1

Una mappa stradale semplificata della Romania, con indicate le distanze stradali in miglia.



3.1 Agenti risolutori di problemi

Immaginiamo un agente che si trova in vacanza in Romania e vuole visitare le zone più belle, migliorare la conoscenza della lingua rumena, fare un po' vita notturna, evitare di ubriacarsi e così via. Il problema decisionale è complesso. Ora supponiamo che l'agente si trovi nella città di Arad e abbia un biglietto aereo non rimborsabile per la partenza da Bucarest il giorno successivo. L'agente osserva la segnaletica stradale e vede che ci sono tre strade che partono da Arad: una verso Sibiu, una verso Timisoara e una verso Zerind. Nessuna di queste è l'obiettivo, perciò l'agente, a meno che non conosca la geografia della Romania, non saprà quale strada seguire.¹

Se l'agente non ha altre informazioni, cioè se l'ambiente è **ignoto**, non può fare altro che eseguire una delle azioni scelte a caso. Questa difficile situazione è descritta nel Capitolo 4. In questo capitolo assumiamo che i nostri agenti possano sempre accedere a informazioni sul mondo, come la mappa della Figura 3.1. Con tali informazioni a disposizione, l'agente può eseguire un processo di risoluzione del problema in quattro fasi.

formulazione dell'obiettivo

- **Formulazione dell'obiettivo:** l'agente adotta l'**obiettivo** di raggiungere Bucarest. Gli obiettivi aiutano a organizzare il comportamento limitando gli scopi e quindi le azioni da considerare.

formulazione del problema

- **Formulazione del problema:** l'agente elabora una descrizione degli stati e delle azioni necessarie per raggiungere l'obiettivo, ovvero un modello astratto della parte del mondo interessata. Per il nostro agente, un buon modello consiste nel considerare le azioni di viaggiare da una città a un'altra città adiacente, perciò l'unico fatto relativo allo stato del mondo che cambierà a causa di un'azione è la città corrente.

¹ Presumiamo che gran parte dei lettori si trovino nelle stesse condizioni del nostro agente e si possano facilmente immaginare altrettanto spediti. Chiediamo scusa ai lettori rumeni, che non potranno trarre vantaggio da questo espediente pedagogico.

- **Ricerca:** prima di effettuare qualsiasi azione nel mondo reale, l'agente simula nel suo modello sequenze di azioni, continuando a cercare finché trova una sequenza che raggiunge l'obiettivo: tale sequenza si chiama **soluzione**. L'agente potrebbe simulare più sequenze che non raggiungono l'obiettivo, ma alla fine troverà una soluzione (come andare da Arad a Sibiu a Fagaras e infine a Bucarest), oppure troverà che non è possibile alcuna soluzione.
 - **Esecuzione:** l'agente ora può eseguire le azioni specificate nella soluzione, una per volta.
- È importante osservare che in un ambiente completamente osservabile, deterministico e noto, la *soluzione di qualsiasi problema è una sequenza fissata di azioni*: guidare fino a Sibiu, poi fino a Fagaras, poi fino a Bucarest. Se il modello è corretto, una volta che l'agente ha trovato una soluzione, può ignorare le sue percezioni mentre esegue le azioni – può chiudere gli occhi, per così dire – dato che ha la garanzia che la soluzione lo condurrà all'obiettivo. Nella teoria del controllo si parla in questo caso di sistema **ad anello aperto**, perché ignorando le percezioni si rompe il ciclo tra agente e ambiente. Se vi è la possibilità che il modello sia errato, o che l'ambiente non sia deterministico, l'agente sarebbe più sicuro usando un approccio **ad anello chiuso** che tiene monitorate le percezioni.

In ambienti parzialmente osservabili o non deterministicci, una soluzione sarebbe una strategia ramificata che consigliasse diverse azioni future in base alle percezioni raccolte. Per esempio, l'agente potrebbe pianificare di viaggiare da Arad a Sibiu ma potrebbe avere bisogno di un piano di riserva nel caso in cui arrivasse a Zerind per caso o trovasse un cartello del tipo “Drum Închis” (strada chiusa).



3.1.1 Problemi di ricerca e soluzioni

Un **problema** di ricerca può essere definito formalmente come segue.

- Un insieme di possibili **stati** in cui può trovarsi l'ambiente. Lo chiamiamo **spazio degli stati**.
- Lo **stato iniziale** in cui si trova l'agente inizialmente. Per esempio: *Arad*.
- Un insieme di uno o più **stati obiettivo**. A volte lo stato obiettivo è uno solo (per esempio *Bucarest*), a volte c'è un piccolo insieme di stati obiettivo alternativi, a volte l'obiettivo è definito da una proprietà che è soddisfatta da molti stati (potenzialmente a un numero infinito). Per esempio, nel mondo dell'aspirapolvere, l'obiettivo potrebbe essere quello che non vi sia sporcizia in alcun posto a prescindere da altri fatti relativi allo stato. Possiamo tenere conto di tutte e tre queste possibilità specificando un metodo È-OBIETTIVO per un problema. In questo capitolo a volte parleremo semplicemente di “obiettivo”, ma quanto detto si applica anche a “uno qualsiasi dei possibili stati obiettivo”.
- Le **azioni** possibili dell'agente. Dato uno stato *s*, AZIONI(*s*) restituisce un insieme finito² di azioni che possono essere eseguite in *s*. Diciamo che ognuna di queste azioni è **applicabile** in *s*. Ecco un esempio:

$$\text{AZIONI}(Arad) = \{\text{VersoSibiu}, \text{VersoTimisoara}, \text{VersoZerind}\}.$$

- Un **modello di transizione** che descrive ciò che fa ogni azione. RISULTATO(*s, a*) restituisce lo stato risultante dall'esecuzione dell'azione *a* nello stato *s*. Per esempio, abbiamo:

$$\text{RISULTATO}(Arad, \text{VersoZerind}) = \text{Zerind}$$

ricerca

soluzione

esecuzione

ad anello aperto

ad anello chiuso

problema

stato

spazio degli stati

stato iniziale

stati obiettivo

azioni

applicabile

modello di transizione

² Per affrontare problemi con un numero infinito di azioni servirebbero tecniche che vanno oltre lo scopo di questo capitolo.

funzione di costo dell'azione

- Una **funzione di costo dell'azione**, denotata da $\text{COSTO-AZIONE}(s, a, s')$ nei programmi o $c(s, a, s')$ nei calcoli matematici, che restituisce il costo numerico di applicare l'azione a nello stato s per raggiungere lo stato s' . Un agente risolutore di problemi dovrebbe usare una funzione di costo che rifletta la sua misura di prestazione; per esempio, nel caso di agenti per la ricerca di cammini, il costo di un'azione potrebbe essere la lunghezza espressa in miglia (come nella Figura 3.1) oppure il tempo richiesto per completare l'azione.

cammino

Una sequenza di azioni forma un **cammino**; una **soluzione** è un cammino che porta dallo stato iniziale a uno stato obiettivo. Assumiamo che i costi delle azioni siano additivi, cioè che il costo totale di un cammino sia la somma dei costi delle singole azioni. Una **soluzione ottima** è, tra tutte le possibili soluzioni, quella che ha il costo minimo. In questo capitolo assumiamo che tutti i costi delle azioni siano positivi, per evitare alcune complicazioni.³

grafo

Lo spazio degli stati può essere rappresentato come un **grafo** in cui i vertici (o nodi) rappresentano gli stati e i collegamenti orientati (archi) tra di essi rappresentano le azioni. La mappa della Romania illustrata nella Figura 3.1 è un grafo di questo tipo, dove ogni strada indica due possibili azioni di spostamento, una per ogni direzione.

3.1.2 La formulazione dei problemi

La nostra formulazione del problema di arrivare a Bucarest è un **modello**, cioè una descrizione matematica astratta, e non qualcosa di reale. Confrontate la semplice descrizione di stato atomica *Arad* con un vero viaggio in macchina attraverso un paese, in cui lo stato del mondo include tante cose: i compagni di viaggio, il programma trasmesso alla radio, il paesaggio fuori dal finestrino, la presenza di poliziotti nelle vicinanze, la distanza per raggiungere la prossima area di servizio, la condizione della strada, le condizioni del tempo, del traffico e così via. Tutte queste considerazioni sono trascurate nel nostro modello perché sono irrilevanti al fine di trovare la strada per Bucarest.

astrazione

Il processo di rimozione dei dettagli da una rappresentazione prende il nome di **astrazione**. Per una buona formulazione del problema serve il giusto livello di dettaglio. Se le azioni fossero al livello di “muovi il piede destro in avanti di un centimetro” o “gira il volante di un grado a sinistra”, l’agente probabilmente non riuscirebbe nemmeno a trovare l’uscita dal parcheggio, figuriamoci arrivare a Bucarest.

livello di astrazione

È possibile essere più precisi riguardo il **livello di astrazione** appropriato? Considerate che agli stati e alle azioni astratte che abbiamo scelto corrispondano grandi insiemi di stati del mondo e di sequenze di azioni dettagliate. Ora considerate una soluzione al problema astratto: per esempio, il cammino da Arad a Sibiu, quindi a Rimnicu Vilcea e da lì a Pitesti, per arrivare infine a Bucarest. Questa soluzione astratta corrisponde a un gran numero di cammini più dettagliati. Per esempio, potremmo guidare con la radio accesa tra Sibiu e Rimnicu Vilcea e poi spegnerla per il resto del viaggio. L’astrazione è *valida* se possiamo espandere ogni so-

³ In qualsiasi problema con un ciclo di costo netto negativo, la soluzione ottima rispetto al costo è quella di percorrere tale ciclo per un numero infinito di volte. Gli algoritmi di Bellman–Ford e Floyd–Warshall (che non tratteremo qui) gestiscono azioni di costo negativo, purché non vi siano cicli negativi. È facile gestire azioni a costo nullo, purché il numero di tali azioni consecutive sia limitato. Per esempio, potremmo avere un robot in cui c’è un costo per gli avanzamenti, ma un costo nullo per ruotare di 90°; gli algoritmi descritti in questo capitolo possono gestire questa situazione purché non siano consentite più di tre rotazioni di 90° consecutive. Ci sono complicazioni anche quando i problemi hanno un numero infinito di azioni di costo arbitrariamente piccolo. Consideriamo una versione del paradosso di Zenone in cui un’azione è arrivare fino a metà strada dall’obiettivo, a un costo pari alla metà della mossa precedente. Il problema non ha soluzioni con un numero finito di azioni, ma per evitare che una ricerca consideri un numero di azioni illimitato senza raggiungere l’obiettivo, possiamo richiedere che tutte le azioni costino almeno ϵ , dove ϵ è un numero positivo piccolo.

luzione astratta in una soluzione nel mondo più dettagliato; una condizione sufficiente è che per ogni stato dettagliato corrispondente a “in Arad” ci sia un cammino dettagliato che porta a qualche stato che corrisponde a “in Sibiu” e così via.⁴ L’astrazione è *utile* se eseguire ogni azione nella soluzione è più facile che nel problema originale; nel nostro caso l’azione “guida da Arad a Sibiu” può essere eseguita da un guidatore di media capacità senza ulteriore ricerca o pianificazione. La scelta di una buona astrazione prevede quindi che si rimuova quanto più dettaglio possibile mantenendo la validità e assicurandosi che le azioni astratte siano facili da eseguire. Se non fosse per la capacità di costruire astrazioni utili, gli agenti intelligenti sarebbero completamente soverchiati dalla complessità del mondo reale.

3.2 Problemi esemplificativi

La risoluzione di problemi è stata applicata a una vasta gamma di ambienti operativi. Ne indicheremo ora qualcuno tra i più noti, distinguendo tra *problemi standardizzati* e *problemi del mondo reale*. Lo scopo di un **problema standardizzato** è illustrare o mettere alla prova diversi metodi di risoluzione di problemi. Può essere descritto in modo preciso e sintetico e quindi è adatto a essere usato come benchmark dai ricercatori per confrontare le prestazioni degli algoritmi. Un **problema del mondo reale**, come quello della navigazione di un robot, è un problema le cui soluzioni sono effettivamente utili alle persone e la cui formulazione è specifica, non standardizzata, perché, per esempio, ciascun robot è dotato di sensori diversi che producono dati diversi.

**problema
standardizzato**

**problema del
mondo reale**

3.2.1 Problemi standardizzati

su griglia

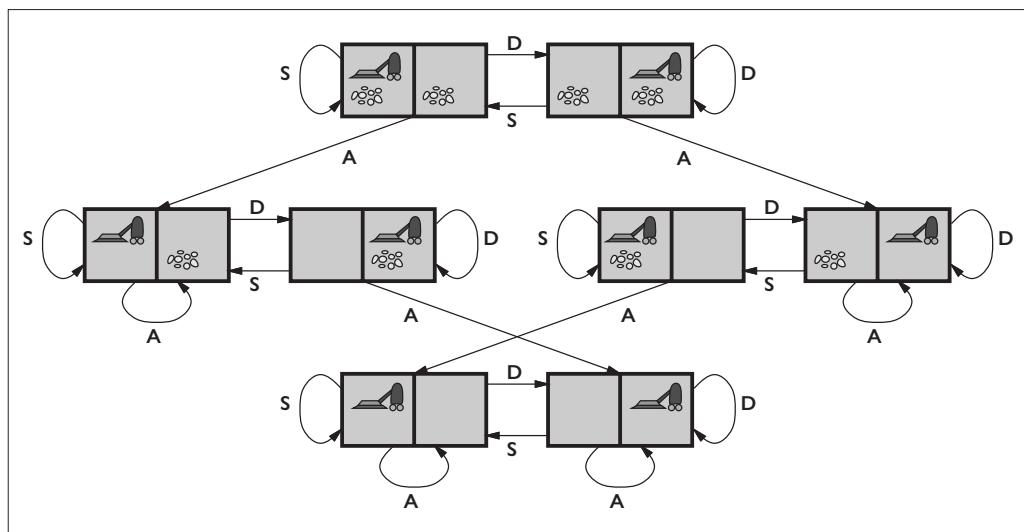
In un problema **su griglia** il mondo è una matrice bidimensionale costituita da un rettangolo di celle quadrate in cui gli agenti possono spostarsi da una cella all’altra. Generalmente l’agente può spostarsi in qualsiasi cella adiacente libera da ostacoli, procedendo in orizzontale o in verticale e in alcuni problemi anche in diagonale. Le celle possono contenere oggetti su cui l’agente può agire in vari modi, come raccoglierli o spingerli. Un muro o altro ostacolo insuperabile in una cella impedisce a un agente di spostarsi in essa. Il **mondo dell’aspirapolvere** presentato nel Paragrafo 2.1 del Capitolo 2 può essere formulato come problema su griglia nel modo seguente.

- **Stati:** uno stato del mondo indica quali oggetti sono in quali celle. Nel mondo dell’aspirapolvere gli oggetti sono l’agente e lo sporco. Nella versione semplice a due riquadri (celle), l’agente può trovarsi in una di due possibili celle, ognuna delle quali può contenere sporco oppure no, quindi ci sono $2 \times 2 \times 2 = 8$ stati (Figura 3.2). In generale, un mondo dell’aspirapolvere con n celle ha $n \times 2^n$ stati.
- **Stato iniziale:** ogni stato può essere designato come stato iniziale.
- **Azioni:** nel mondo a due celle abbiamo definito tre azioni: *Sinistra*, *Destra* e *Aspira*. In un mondo bidimensionale a più celle ci servirebbero più azioni di spostamento. Potremmo aggiungere *Su* e *Giù*, ottenendo così quattro azioni di movimento **assolute**, oppure potremmo passare ad **azioni egocentriche**, definite in riferimento al punto di vista dell’agente, per esempio *Avanti*, *Indietro*, *GiraDestra* e *GiraSinistra*.
- **Modello di transizione:** l’azione *Aspira* rimuove lo sporco dalla cella dove si trova l’agente; *Avanti* muove l’agente di una cella nella direzione verso cui è orientato, a meno che non abbia davanti un muro, nel qual caso l’azione non ha effetto. *Indietro* sposta l’agente

⁴ Si veda il Paragrafo 11.4.

Figura 3.2

Il grafo che rappresenta lo spazio degli stati per il mondo dell'aspirapolvere a due celle. Ci sono 8 stati e 3 azioni per ciascuno stato: S = Sinistra, D = Destra, A = Aspira.



nella direzione opposta, mentre *GiraDestra* e *GiraSinistra* cambiano di 90° la direzione in cui è orientato l'agente.

- **Stati obiettivo:** gli stati in cui ogni cella è pulita.
- **Costo di azione:** ogni azione costa 1.

rompicapo sokoban

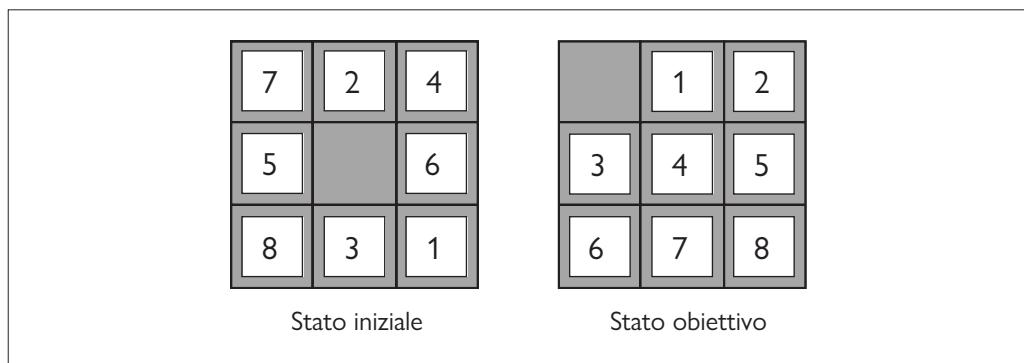
Un altro tipo di problema su griglia è il **rompicapo sokoban**, in cui l'obiettivo dell'agente è spingere un certo numero di scatole sparse sulla griglia verso posizioni stabilite. Può esserci al più una scatola per cella. Quando un agente si muove in avanti verso una cella contenente una scatola e c'è una cella vuota dall'altra parte della scatola, si muovono in avanti sia la scatola che l'agente. L'agente non può spingere una scatola verso un'altra scatola o contro un muro. Per un mondo con n celle senza ostacoli e b scatole, ci sono $n \times n! / (b!(n - b)!)$ stati; per esempio, su una griglia 8×8 con una dozzina di scatole, ci sono oltre 200.000 miliardi di stati.

rompicapo a tasselli mobili

In un **rompicapo a tasselli mobili**, un certo numero di tasselli (chiamati anche blocchi o pezzi) sono disposti in una griglia con uno o più spazi vuoti in modo tale che qualcuno dei tasselli possa scorrere in uno spazio vuoto. Una variante è il rompicapo dell'ora di punta, in cui automobili e camion scorrono in una griglia 6×6 nel tentativo di liberare un'auto dal traffico intasato. Le varianti forse più note sono il **rompicapo a 8 tasselli** (Figura 3.3), costituito da una griglia 3×3 con otto tasselli numerati e uno spazio vuoto, e il **rompicapo a 15 tasselli** (gioco del 15) in cui c'è una griglia 4×4 . Lo scopo è raggiungere uno stato obiettivo

Figura 3.3

Una tipica istanza del rompicapo a 8 tasselli.



specificato, come quello mostrato a destra nella Figura 3.3. La formulazione standard del rompicapo a 8 tasselli è la seguente.

- **Stati:** una descrizione di stato specifica la posizione di ognuno degli otto tasselli.
- **Stato iniziale:** ogni stato può essere designato come stato iniziale. Notate che, qualsiasi sia lo stato obiettivo, questo potrà essere raggiunto da esattamente metà dei possibili stati iniziali (Esercizio 3.PART).
- **Azioni:** mentre nel mondo fisico sono i tasselli a scorrere, il modo più semplice per descrivere un'azione è quello di pensarla come se fosse lo spazio vuoto a muoversi a *Sinistra*, a *Destra*, *Su* o *Giù*. Se lo spazio vuoto si trova su un bordo o su un angolo, non tutte le azioni sono applicabili.
- **Modello di transizione:** fa corrispondere a uno stato e un'azione lo stato risultante; per esempio, se applichiamo *Sinistra* allo stato iniziale della Figura 3.3, lo stato risultante prevede il tassello 5 e lo spazio vuoto scambiati di posto.
- **Stato obiettivo:** anche se ogni stato potrebbe essere quello obiettivo, generalmente specifichiamo uno stato con i numeri ordinati, come nella Figura 3.3.
- **Costo di azione:** ogni azione costa 1.

Notate che ogni formulazione di problema comporta delle astrazioni. Nel rompicapo a 8 tasselli le azioni sono astratte ai loro stati di partenza e di arrivo, ignorando le posizioni intermedie in cui il tassello scivola. Mediante l'astrazione abbiamo eliminato azioni come scuotere la plancia quando i pezzi si incastrano, e non abbiamo ammesso la possibilità di tirare fuori tutti i pezzi con un coltellino e poi rimetterli dentro. Ciò che rimane è la descrizione delle regole del rompicapo, senza alcun riferimento ai dettagli della manipolazione fisica.

L'ultimo problema standardizzato che descriviamo si deve a Donald Knuth (1964) e illustra come possano nascere spazi degli stati infiniti. Knuth ipotizzò che, iniziando con il numero 4, una sequenza di operazioni di radice quadrata, parte intera e fattoriale potesse raggiungere qualsiasi intero positivo desiderato. Per esempio, possiamo raggiungere 5 da 4 nel modo seguente:

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \rfloor = 5$$

La definizione del problema è semplice:

- **Stati:** numeri reali positivi.
- **Stato iniziale:** 4.
- **Azioni:** applicare operazioni di radice quadrata, parte intera o fattoriale (fattoriale soltanto per numeri interi).
- **Modello di transizione:** dato dalle definizioni matematiche delle operazioni.
- **Stato obiettivo:** l'intero positivo desiderato.

Lo spazio degli stati per questo problema è infinito: per qualsiasi intero maggiore di 2 l'operatore di fattoriale restituirà sempre un intero più grande. Il problema è interessante perché esamina numeri molto grandi: il cammino più breve per arrivare a 5 passa da $(4!)! = 620.448.401.733.239.439.360.000$. Spazi degli stati infiniti si presentano frequentemente nelle attività che coinvolgono la generazione di espressioni matematiche, circuiti, dimostrazioni, programmi e altri oggetti definiti in modo ricorsivo.

3.2.2 Problemi reali

Abbiamo già visto come un **problema di ricerca dell'itinerario** sia definito specificando una serie di locazioni e di transizioni lungo gli archi che le collegano. Gli algoritmi che trovano itinerari sono usati in una varietà di applicazioni. Alcuni, come i siti web e i navigatori per

le automobili che forniscono indicazioni di percorso, si possono considerare estensioni relativamente dirette dell'esempio della Romania (le principali complicazioni sono i costi che variano a causa dei ritardi dovuti al traffico e la necessità di ricalcolare i percorsi a causa dei lavori stradali). Altri, come l'instradamento di flussi video nelle reti di computer, la pianificazione delle operazioni militari e i sistemi di supporto per la generazione di pacchetti di viaggi aerei, sono molto più difficili da specificare. Consideriamo i problemi di viaggio aereo che devono essere risolti da un sito web per la pianificazione dei viaggi.

- **Stati:** ognuno ovviamente comprende una posizione (per esempio un aeroporto) e l'ora corrente. Inoltre, poiché il costo di un'azione (un tratto di volo) può dipendere da eventuali tratte precedenti, dalle loro tariffe e dal loro stato di tratte nazionali o internazionali, lo stato deve registrare altre informazioni su questi aspetti "storici".
- **Stato iniziale:** l'aeroporto da dove parte l'utente.
- **Azioni:** prendere un volo dalla posizione corrente, in una classe qualsiasi, partendo dopo l'ora corrente, lasciando tempo sufficiente per il trasferimento all'interno dell'aeroporto, se necessario.
- **Modello di transizione:** lo stato risultante dal prendere un volo avrà come nuova posizione la destinazione del volo e come ora corrente quella di arrivo del volo.
- **Stato obiettivo:** una città di destinazione. A volte l'obiettivo può essere più complesso, come "arrivare a destinazione su un volo diretto".
- **Costo di azione:** una combinazione di costo monetario, tempi di attesa, durata dei voli, procedure di dogana, qualità dei posti a sedere, ora del giorno, tipo di aereo, punti per programmi frequent flyer e così via.

I sistemi commerciali di supporto per la generazione di pacchetti di viaggi aerei usano una formulazione del problema simile a questa, ulteriormente complicata dalla necessità di gestire le complesse politiche di prezzo delle compagnie aeree. Ogni viaggiatore esperto sa, tuttavia, che non tutti i voli vanno come previsto. Un sistema davvero buono dovrebbe prevedere dei piani di riserva, per esempio per i casi di ritardo del volo e perdita di coincidenze.

problema di viaggio

problema del commesso viaggiatore

configurazione VLSI

I **problematici di viaggio** descrivono una serie di località che devono essere visitate, anziché una singola destinazione obiettivo. Il **problema del commesso viaggiatore** (TSP, *traveling salesperson problem*) è un problema di viaggio in cui ogni città presente su una mappa va visitata. Lo scopo è trovare un itinerario con costo $< C$ (o, nella versione di ottimizzazione, trovare l'itinerario con il minimo costo possibile). Sono stati compiuti grandi sforzi per migliorare gli algoritmi per il problema del commesso viaggiatore. Tali algoritmi possono anche essere estesi per gestire flotte di veicoli. Per esempio, un algoritmo di ricerca e ottimizzazione per pianificare gli itinerari degli autobus scolastici a Boston ha consentito di risparmiare 5 milioni di dollari e di ridurre il traffico e l'inquinamento dell'aria, oltre al tempo perso da autisti e studenti (Bertsimas *et al.*, 2019). Oltre a pianificare itinerari di viaggio, gli algoritmi di ricerca sono stati usati per attività quali la pianificazione dei movimenti per i macchinari di produzione di schede elettroniche e per i macchinari che gestiscono lo stoccaggio di prodotti nei magazzini.

Un problema di **configurazione VLSI** richiede che vengano posizionati al meglio i milioni di componenti e di connessioni che formano un chip in modo da minimizzare l'area del chip stesso, i ritardi dei circuiti e i problemi elettrici, massimizzando così la resa della produzione. Il problema di configurazione viene subito dopo la fase di progettazione logica, e solitamente è diviso in due parti: **configurazione delle celle** e **inistradamento dei canali**. Nella configurazione delle celle, i componenti di basso livello sono raggruppati in celle, ognuna delle quali svolge una funzione ben definita. Ogni cella ha una forma e dimensione prefissata e richiede un certo numero di connessioni con le altre. L'obiettivo è dislocare le celle sul chip in modo che non si sovrappongano e che vi sia spazio per le piste di collegamento necessarie. L'inistradamento

dei canali serve a definire il cammino preciso di ogni pista negli spazi tra le celle. Questi problemi di ricerca sono davvero complessi, ma meritano decisamente di essere risolti.

La **navigazione dei robot** è una generalizzazione del problema di ricerca dell'itinerario che abbiamo descritto precedentemente. Invece di seguire itinerari distinti (come le strade in Romania), un robot si può vagare nello spazio creando da sé i propri itinerari. Per un robot di forma circolare che si muove su una superficie piana, lo spazio è essenzialmente bidimensionale. Quando il robot è dotato di braccia e gambe che devono essere controllate anch'esse, lo spazio di ricerca diventa multidimensionale, con una dimensione per ogni angolo controllabile da un giunto. In questo caso sono richieste tecniche avanzate soltanto per rendere finito lo spazio di ricerca essenzialmente continuo (si veda il Capitolo 26 del Volume 2). Oltre alla complessità intrinseca nel problema, i robot reali devono anche gestire gli errori nelle letture dei sensori e nei controlli dei motori, oltre all'osservabilità parziale e ad altri agenti che potrebbero alterare l'ambiente.

Sequenze di montaggio automatico di oggetti complessi (come i motori elettrici) da parte di un robot sono prassi industriali standard fin dagli anni 1970. Gli algoritmi prima trovano una sequenza di montaggio ammissibile e poi lavorano per ottimizzare il processo. Minimizzando il lavoro manuale umano necessario nella catena di montaggio, si possono ottenere notevoli risparmi di tempi e costi. In questi problemi lo scopo è trovare l'ordine in cui assemblare le varie parti dell'oggetto: se questo è sbagliato, a un certo punto non sarà possibile aggiungere un pezzo senza smontare una parte del lavoro già svolto. Controllare che una determinata azione nella sequenza sia ammissibile è un difficile problema di ricerca geometrico, molto simile alla navigazione dei robot. Di conseguenza, la generazione di azioni legali è la parte più costosa della costruzione di una sequenza di montaggio. Ogni algoritmo, se vuole essere applicabile nella pratica, deve evitare di esplorare lo spazio degli stati nella sua interezza e considerarne solo una frazione molto piccola. Un importante problema di montaggio è la **progettazione di proteine**, il cui obiettivo è trovare una sequenza di aminoacidi in grado di ripiegarsi in una proteina tridimensionale con le caratteristiche ideali per curare una certa malattia.

navigazione
dei robot

sequenza
di montaggio
automatico

progettazione
di proteine

3.3 Algoritmi di ricerca

Un **algoritmo di ricerca** riceve in input un problema di ricerca e restituisce una soluzione o un'indicazione di fallimento. In questo capitolo consideriamo algoritmi che sovrappongono un **albero di ricerca** al grafo dello spazio degli stati, formando vari cammini a partire dallo stato iniziale e cercando di trovarne uno che raggiunga uno stato obiettivo. Ciascun **nodo** nell'albero di ricerca corrisponde a uno stato nello spazio degli stati e i rami dell'albero di ricerca corrispondono ad azioni. La radice dell'albero corrisponde allo stato iniziale del problema.

È importante comprendere la distinzione tra spazio degli stati e albero di ricerca. Lo spazio degli stati descrive l'insieme (eventualmente infinito) degli stati nel mondo e le azioni che consentono le transizioni da uno stato a un altro. L'albero di ricerca descrive cammini tra questi stati per raggiungere l'obiettivo. Nell'albero di ricerca possono esserci più cammini per raggiungere qualsiasi stato (e quindi anche più nodi corrispondenti allo stesso stato), ma per ogni nodo dell'albero c'è un cammino univoco per tornare alla radice (come avviene in tutti gli alberi).

La Figura 3.4 mostra i primi passi nella ricerca di un cammino che porti da Arad a Bucharest. Il nodo radice dell'albero di ricerca corrisponde allo stato iniziale, *Arad*. Possiamo **espandere** il nodo, considerando le **AZIONI** disponibili per quello stato, usando la funzione **RISULTATO** per vedere dove portano tali azioni e **generare** un nuovo nodo (chiamato **nodo figlio** o **nodo successore**) per ognuno degli stati risultanti. Ogni nodo figlio ha *Arad* come **nodo padre**.

algoritmo di ricerca

nodo

espandere
generare
nodo figlio
nodo successore
nodo padre

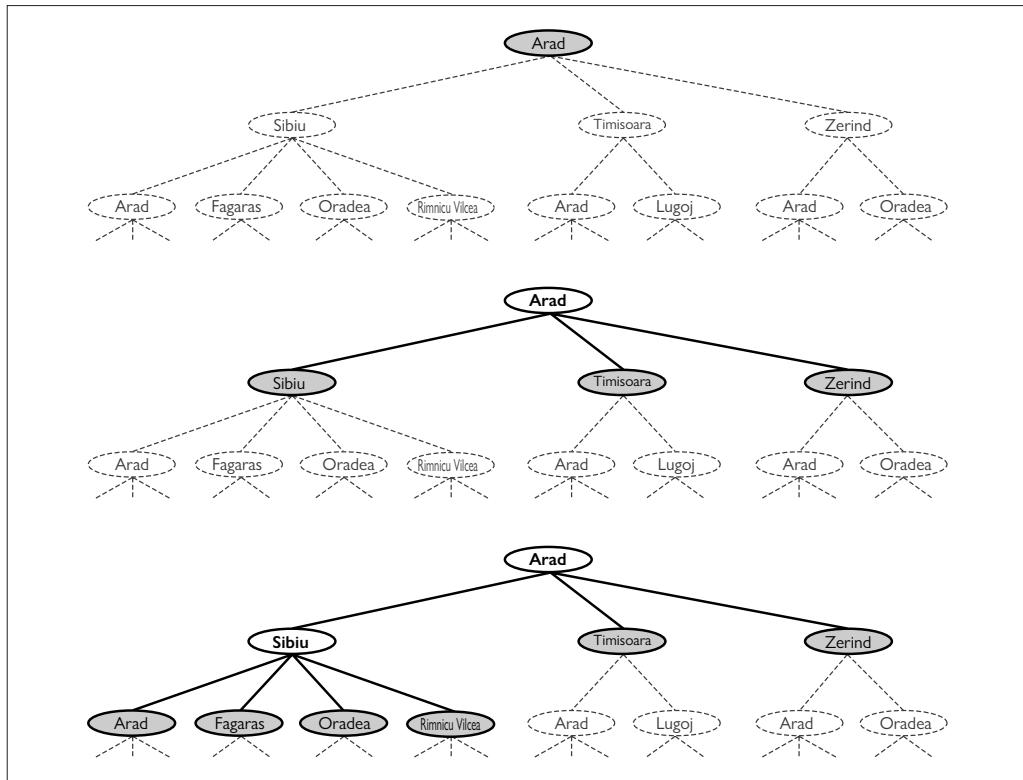


Figura 3.4 Tre alberi di ricerca parziali per trovare l’itinerario da Arad a Bucarest. I nodi già espansi sono in bianco con caratteri in grassetto; quelli nella frontiera che sono stati generati ma non ancora espansi sono in grigio; gli stati corrispondenti a questi due tipi di nodi si dicono raggiunti. I nodi che potrebbero essere generati al passo successivo sono indicati con un tratteggio più chiaro. Notate che nell’albero più in basso c’è un ciclo da Arad a Sibiu ad Arad; quello non può essere un cammino ottimale, perciò la ricerca non dovrebbe continuare da lì.

frontiera
raggiunto
separatore

Ora dobbiamo scegliere quale dei nodi figli considerare. Questa è l’essenza della ricerca: seguire un’opzione e mettere per il momento da parte le altre, pronti a riprenderle in seguito. Supponiamo di scegliere di espandere prima Sibiu. La Figura 3.4 (in basso) mostra il risultato: un insieme di 6 nodi non espansi (evidenziati in grassetto). Chiamiamo l’insieme di questi nodi **frontiera** dell’albero di ricerca, e diciamo che ogni stato per cui vi è un nodo generato è stato **raggiunto** (indipendentemente dal fatto che tale nodo sia stato espanso o meno).⁵ La Figura 3.5 mostra l’albero di ricerca sovrapposto al grafo dello spazio degli stati.

Notate che la frontiera **separa** due regioni del grafo dello spazio degli stati: una regione interna in cui ogni stato è stato espanso e una esterna di stati che non sono ancora stati raggiunti. Questa proprietà è illustrata nella Figura 3.6.

⁵ Alcuni autori chiamano la frontiera **lista aperta**, che tuttavia è meno evocativo dal punto di vista geografico e meno appropriato dal punto di vista computazionale, perché in questo caso una coda è più efficiente di una lista. Questi autori utilizzano il termine **lista chiusa** per indicare l’insieme dei nodi precedentemente espansi, che nella nostra terminologia sarebbe quello dei nodi *raggiunti* meno la *frontiera*.

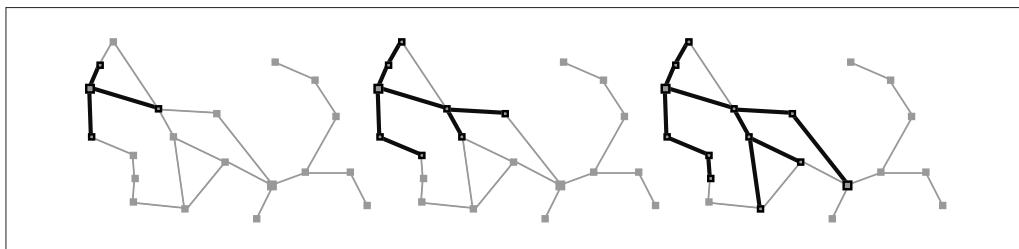


Figura 3.5 Una sequenza di alberi di ricerca generati da una ricerca su grafo per il problema riferito alla Romania descritto nella Figura 3.1. In ciascun passaggio abbiamo espanso ogni nodo sulla frontiera, estendendo ogni cammino con tutte le azioni applicabili che non portano a uno stato già raggiunto in precedenza. Notate che al terzo passo la città più in alto (Oradea) ha due successori entrambi già raggiunti da altri cammini, perciò non è possibile estendere cammini da Oradea.

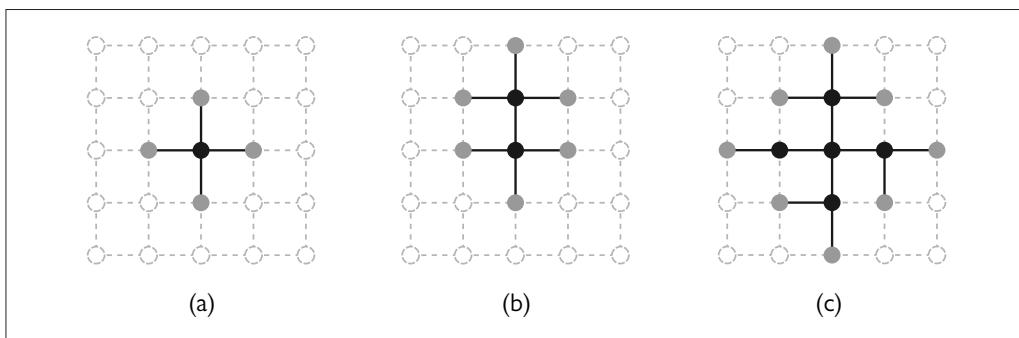


Figura 3.6 La proprietà di separazione della ricerca su grafo, illustrata con un problema su griglia rettangolare. La frontiera (grigio) separa l'interno (nero) dall'esterno (tratteggio chiaro). La frontiera è l'insieme dei nodi (e corrispondenti stati) che sono stati raggiunti ma non ancora espansi; l'interno è l'insieme dei nodi (e corrispondenti stati) che sono stati espansi; l'esterno è l'insieme degli stati che non sono stati raggiunti. In (a), soltanto il nodo radice è stato espanso. In (b) è stato espanso il nodo più in alto sulla frontiera. In (c) sono stati espansi i rimanenti nodi successori della radice, procedendo in senso orario.

3.3.1 Ricerca best-first

Come decidiamo quale nodo della frontiera espandere? Un approccio molto generale è quello della **ricerca best-first** (letteralmente “prima il migliore”), in cui scegliamo un nodo n che corrisponde al valore minimo di una **funzione di valutazione** $f(n)$. L’algoritmo è mostrato nella Figura 3.7. A ogni iterazione scegliamo un nodo sulla frontiera in cui $f(n)$ ha valore minimo e lo restituiamo se il suo stato è uno stato obiettivo, altrimenti applichiamo ESPANDI per generare nodi figli. Ogni nodo figlio viene aggiunto alla frontiera se non è stato raggiunto in precedenza, o viene aggiunto di nuovo se ora è stato raggiunto con un cammino di costo inferiore ai precedenti. L’algoritmo restituisce un’indicazione di fallimento oppure un nodo che rappresenta un cammino che porta a un obiettivo. Utilizzando funzioni $f(n)$ diverse otteniamo algoritmi specifici diversi, che tratteremo in questo capitolo.

ricerca best-first
funzione di valutazione

3.3.2 Strutture dati per la ricerca

Gli algoritmi di ricerca richiedono una struttura dati per tenere traccia dell’albero di ricerca. Un **nodo** nell’albero di ricerca è rappresentato da una struttura dati con quattro componenti:

- $n.\text{STATO}$: lo stato a cui corrisponde il nodo;
- $n.\text{PADRE}$: il nodo dell’albero di ricerca che ha generato il nodo corrente;

```

function RICERCA-BEST-FIRST(problema, f) returns un nodo soluzione o fallimento
  nodo  $\leftarrow$  NODO(STATO=problema.STATOINIZIALE)
  frontiera  $\leftarrow$  una coda con priorità ordinata in base a f, con nodo come elemento iniziale
  raggiunti  $\leftarrow$  una tabella di lookup, con un elemento con chiave problema.STATOINIZIALE e valore nodo
  while not VUOTA?(frontiera) do
    nodo  $\leftarrow$  POP(frontiera)
    if problema.È-OBIETTIVO(nodo.STATO) then return nodo
    for each figlio in ESPANDI(problema, nodo) do
      s  $\leftarrow$  figlio.STATO
      if s non è in raggiunti or figlio.COSTO-CAMMINO < raggiunti[s].COSTO-CAMMINO then
        raggiunti[s]  $\leftarrow$  figlio
        aggiungi figlio a frontiera
  return fallimento

function ESPANDI(problema, nodo) yields nodi
  s  $\leftarrow$  nodo.STATO
  for each azione in problema.AZIONI(s) do
    s'  $\leftarrow$  problema.RISULTATO(s, azione)
    costo  $\leftarrow$  nodo.COSTO-CAMMINO + problema.COSTO-AZIONE(s, azione, s')
    yield NODO(STATO=s', PADRE=nodo, AZIONE=azione, COSTO-CAMMINO=costo)

```

Figura 3.7 L’algoritmo di ricerca best-first e la funzione per espandere un nodo. Le strutture dati usate qui sono descritte nel Paragrafo 3.3.2. Si veda l’Appendice B per **yield**.

- *n.AZIONE*: l’azione applicata allo stato del padre per generare il nodo corrente;
- *n.COSTO-CAMMINO*: il costo totale del cammino che va dallo stato iniziale al nodo corrente.
Nelle formule matematiche utilizziamo $g(nodo)$ come sinonimo di COSTO-CAMMINO.

Seguendo i puntatori PADRE a partire da un nodo possiamo risalire agli stati e alle azioni lungo il cammino fino a tale nodo. Procedendo in questo modo a partire da un nodo obiettivo troviamo la soluzione.

coda Ci serve una struttura dati per memorizzare la **frontiera**. La scelta più appropriata è una **coda**, perché le operazioni svolte su una frontiera sono:

- VUOTA?(*frontiera*) restituisce true (vero) solo se non ci sono più nodi nella frontiera.
- POP(*frontiera*) rimuove il nodo in cima alla frontiera e lo restituisce.
- TOP(*frontiera*) restituisce (senza rimuoverlo) il nodo in cima alla frontiera.
- AGGIUNGI(*nodo, frontiera*) inserisce un nodo al posto appropriato nella coda.

Negli algoritmi di ricerca si utilizzano tre tipi di code:

- | | |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| coda con priorità | • la coda con priorità , in cui viene estratto prima il nodo di costo minimo in base a una funzione di valutazione <i>f</i> . Questo tipo di coda è usato nella ricerca best-first; |
| coda FIFO | • la coda FIFO o <i>First-In, First-Out</i> (il primo che entra è il primo a uscire), in cui viene estratto prima il nodo che è stato aggiunto alla coda per primo. Vedremo che questo tipo di coda è usato nella ricerca in ampiezza; |
| coda LIFO
stack | • la coda LIFO o <i>Last-In, First-Out</i> (l’ultimo che entra è il primo a uscire), detta anche stack , in cui viene estratto il nodo che è stato aggiunto alla coda per ultimo. Vedremo che questo tipo di coda è usato nella ricerca in profondità. |

Gli stati raggiunti possono essere memorizzati come tabella di lookup (per esempio una tabella hash) in cui ogni chiave è uno stato e ogni valore è il nodo per tale stato.

3.3.3 Cammini ridondanti

L'albero di ricerca illustrato nella Figura 3.4 (in basso) include un cammino da Arad a Sibiu e poi ancora ad Arad. Diciamo che **Arad** è uno **stato ripetuto** nell'albero di ricerca, generato in questo caso da un **ciclo** (o **cammino ciclico**). Perciò, anche se lo spazio degli stati ha soltanto 20 stati, l'albero di ricerca completo è *infinito* perché non vi è limite al numero di volte per cui è possibile percorrere un ciclo.

Un ciclo è un caso particolare di **cammino ridondante**. Per esempio, possiamo arrivare a Sibiu attraverso il cammino Arad–Sibiu (140 miglia) o il cammino Arad–Zerind–Oradea–Sibiu (297 miglia): il secondo cammino è ridondante, rappresenta semplicemente una via peggiore per arrivare allo stesso stato, e non va considerato nella ricerca di cammini ottimi.

Consideriamo un agente in un mondo costituito da una griglia 10×10 , che è in grado di andare in uno qualsiasi di 8 celle adiacenti. Se non ci sono ostacoli, l'agente può raggiungere una qualsiasi delle cento celle della griglia in al più 9 mosse. Ma il numero dei cammini di lunghezza 9 è quasi 8^9 (un po' meno a causa dei lati della griglia), più di 100 milioni. In altre parole, una cella può essere raggiunta in media da oltre un milione di cammini ridondanti di lunghezza 9, e se eliminiamo i cammini ridondanti possiamo completare una ricerca circa un milione di volte più velocemente. Dice un detto: *gli algoritmi che non sanno ricordare il passato sono condannati a ripeterlo*. Questo problema può essere affrontato con tre approcci.

Il primo approccio consiste nel ricordare tutti gli stati precedentemente raggiunti (come avviene nella ricerca best-first), il che ci consente di individuare tutti i cammini ridondanti e mantenere soltanto il cammino migliore per raggiungere ogni stato. Questo approccio è adatto per spazi degli stati in cui vi sono molti cammini ridondanti, ed è la scelta preferita quando c'è spazio di memoria sufficiente per contenere la tabella degli stati raggiunti.

Il secondo approccio consiste nel non preoccuparsi di ripetere il passato. Esistono alcune formulazioni di problemi in cui è raro o impossibile che due cammini conducano allo stesso stato. Un esempio è quello di un problema di montaggio in cui ogni azione aggiunge un componente a una struttura in costruzione ed esiste un ordinamento per cui è possibile aggiungere *A* e poi *B* ma non *B* e poi *A*. Per questi problemi possiamo risparmiare spazio in memoria se *non* tracciamo gli stati raggiunti e non controlliamo la presenza di cammini ridondanti. Un algoritmo di ricerca è detto di **ricerca su grafo** se controlla la presenza di cammini ridondanti e **ricerca ad albero**⁶ se non esegue tale controllo. L'algoritmo RICERCA-BEST-FIRST della Figura 3.7 è di ricerca su grafo: se rimuoviamo tutti i riferimenti a *raggiunti* otteniamo una ricerca ad albero che utilizza meno memoria ma esamina i cammini ridondanti che portano allo stesso stato, perciò sarà più lenta.

Il terzo approccio è un compromesso: controlliamo la presenza di cammini ciclici, ma non di cammini ridondanti. Poiché ogni nodo ha una catena di puntatori padre, possiamo controllare la presenza di cammini ciclici senza la necessità di usare memoria aggiuntiva risalendo la catena dei padri in modo da verificare se lo stato al termine del cammino è già apparsso precedentemente. Alcune implementazioni seguono comunque tutta la catena e poi eliminano tutti i cammini ciclici, mentre altre seguono solo pochi collegamenti (per esempio fino al padre, nonno e bisnonno) e quindi richiedono una quantità di tempo costante, ma

stato ripetuto
ciclo
cammino ciclico

cammino
ridondante



ricerca su grafo
ricerca ad albero

⁶ Il termine “ricerca ad albero” è usato perché lo spazio degli stati è sempre lo stesso grafo indipendentemente dal modo in cui effettuiamo la ricerca; sceglieremo semplicemente di considerarlo *come se fosse un albero*, con un solo cammino che risale da ciascun nodo verso la radice.

eliminano solo i cammini ciclici brevi (affidandosi ad altri meccanismi per i cammini ciclici più lunghi).

3.3.4 Misurare le prestazioni nella risoluzione di problemi

Prima di esaminare la progettazione di vari algoritmi di ricerca, consideriamo i criteri usati per sceglierli. Possiamo valutare le prestazioni di un algoritmo secondo quattro parametri.

completezza

- **Completezza:** l'algoritmo garantisce di trovare una soluzione, se questa esiste, e di riportare correttamente il fallimento, se la soluzione non esiste?

ottimalità rispetto al costo

- **Ottimalità rispetto al costo:** trova la soluzione con il costo di cammino minimo fra tutte?⁷

complessità temporale

- **Complessità temporale:** quanto tempo impiega a trovare una soluzione? Il tempo può essere misurato in secondi o, in modo più astratto, in base al numero di stati e azioni considerato.

complessità spaziale

- **Complessità spaziale:** di quanta memoria necessita per effettuare la ricerca?

Per comprendere la completezza, consideriamo un problema di ricerca con un singolo obiettivo che può essere ovunque nello spazio degli stati; un algoritmo completo, quindi, deve essere in grado di esplorare sistematicamente ogni stato che sia raggiungibile a partire dallo stato iniziale. In spazi degli stati finiti è facile: basta che manteniamo traccia dei cammini scartando quelli ciclici (per esempio da Arad a Sibiu ad Arad) e alla fine raggiungeremo ogni stato raggiungibile.

In spazi degli stati infiniti, invece, serve maggiore attenzione. Per esempio, un algoritmo che applicasse ripetutamente l'operatore “fattoriale” nel “problema del numero 4” di Knuth seguirebbe un cammino infinito da 4 a $4!$ a $(4!)!$ e così via. Analogamente, su una griglia infinita senza ostacoli, muoversi ripetutamente in avanti lungo una linea retta porta a seguire un cammino infinito di nuovi stati. In entrambi i casi l'algoritmo non torna mai a uno stato che ha già raggiunto prima, ma è incompleto perché ci sono porzioni dello spazio degli stati che non vengono mai raggiunte.

sistemico

Per essere completo, un algoritmo di ricerca deve procedere in modo **sistemico** nell'esplorazione di uno spazio degli stati infinito, assicurandosi di poter raggiungere qualsiasi stato collegato a quello iniziale. Per esempio, sulla griglia infinita, un tipo di ricerca sistematica è un cammino a spirale che copre tutte le celle che si trovano a s passi di distanza dall'origine prima di spostarsi nelle celle che si trovano a $s + 1$ passi di distanza. Sfortunatamente, in uno spazio degli stati infinito senza soluzione, un algoritmo corretto deve continuare a cercare per sempre; non può terminare perché non può sapere se il prossimo stato sarà un obiettivo.

La complessità spaziale e quella temporale sono sempre considerate in rapporto a una misura della difficoltà del problema. Nell'informatica teorica la misura tipica è la dimensione del grafo degli stati, $|V| + |E|$, dove V è l'insieme dei vertici (corrispondenti agli stati) del grafo ed E è l'insieme dei collegamenti o archi (coppie distinte stato/azione). Questo è appropriato quando il grafo è una struttura dati esplicita, come la mappa della Romania. Tuttavia, in molti problemi di IA il grafo è rappresentato soltanto *implicitamente* dallo stato iniziale, dalle azioni e dal modello di transizione. Per uno spazio degli stati implicito, la complessità si può misurare in termini di d , la **profondità (depth)**, ovvero il numero di azioni di una soluzione ottima; m , il massimo numero di azioni in qualsiasi cammino; e b , il **fattore di ramificazione (branching factor)** o numero di successori di un nodo che devono essere considerati.

⁷ Alcuni autori utilizzano il termine “ammissibilità” per indicare la proprietà di trovare la soluzione di costo minimo, mentre alcuni usano “ottimalità” che però può essere confuso con altri tipi di ottimalità.

3.4 Strategie di ricerca non informata

Un algoritmo di ricerca non informata non riceve alcuna informazione su quanto uno stato sia vicino all'obiettivo. Per esempio, consideriamo il nostro agente che si trova ad Arad con l'obiettivo di raggiungere Bucarest. Un agente non informato privo di conoscenza della geografia romena non ha modo di sapere se sia meglio come primo passo andare a Zerind o Sibiu. Al contrario, un agente informato (Paragrafo 3.5) che conosce la posizione di ogni città sa che Sibiu è molto più vicina a Bucarest e quindi ha più probabilità di trovarsi sul cammino più breve.

3.4.1 Ricerca in ampiezza

Quando tutte le azioni hanno lo stesso costo, una strategia appropriata è la **ricerca in ampiezza**, in cui si espande prima il nodo radice, poi tutti i suoi successori, poi i successori di questi ultimi, e così via. Si tratta di una ricerca sistematica, quindi completa anche su spazi degli stati infiniti.

Potremmo implementare la ricerca in ampiezza come chiamata di RICERCA-BEST-FIRST con la funzione di valutazione $f(n)$ è uguale alla profondità del nodo, cioè al numero di azioni necessarie per raggiungerlo. Tuttavia, con alcune trucchi possiamo ottenere maggiore efficienza. Una coda FIFO (*first-in-first-out*) risulterà più veloce di una coda con priorità e ci fornirà l'ordine corretto dei nodi: i nuovi nodi (che sono sempre a profondità maggiore rispetto ai loro padri) vanno in fondo alla coda e quelli vecchi, a profondità minore, vengono espansi per primi. Inoltre, *raggiunti* può essere un insieme di stati, anziché una corrispondenza da stati a nodi, perché una volta raggiunto uno stato, non possiamo più trovare un cammino migliore per raggiungerlo. Questo significa anche che possiamo effettuare un **test obiettivo anticipato**, controllando se un nodo è una soluzione non appena viene *generato*, invece di effettuare il **test obiettivo a posteriori** come nella ricerca best-first, aspettando che un nodo sia estratto dalla coda. La Figura 3.8 mostra il progredire di una ricerca in ampiezza su un albero binario e la Figura 3.9 riporta l'algoritmo con i miglioramenti descritti per una maggiore efficienza.

La ricerca in ampiezza trova sempre una soluzione con un numero minimo di azioni, perché quando genera nodi di profondità d , ha già generato tutti i nodi di profondità $d - 1$, perciò se uno di essi fosse una soluzione, sarebbe stato trovato. Ciò significa che è ottimale rispetto al costo per problemi in cui tutte le azioni hanno lo stesso costo, ma non per problemi che non hanno tale proprietà. È completa in ogni caso. In termini di tempo e spazio, supponiamo di effettuare una ricerca in un albero uniforme dove ogni stato ha b successori; la radice dell'albero di ricerca genera b nodi, ognuno dei quali genera altri b nodi, per un totale di b^2 al secondo livello. Ognuno di questi genera altri b nodi, per un totale di b^3 al terzo livello, e così via. Ora supponiamo che la soluzione sia a profondità d . Allora il numero totale di nodi generati è:

$$1 + b + b^2 + b^3 + \dots + b^d = O(b^d).$$

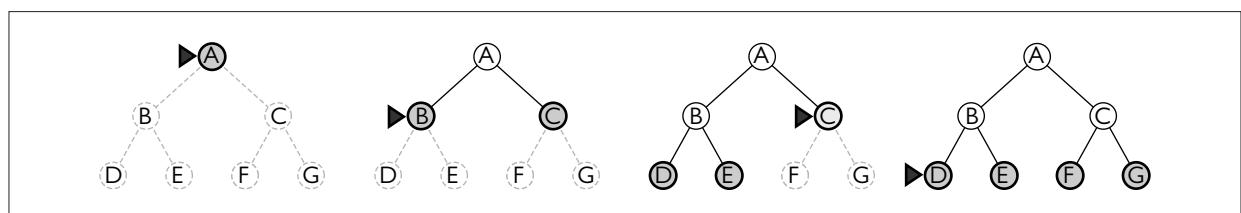


Figura 3.8 Ricerca in ampiezza su un semplice albero binario. A ogni passo, il prossimo nodo da espandere è indicato dal triangolino.

ricerca in ampiezza

test obiettivo anticipato

test obiettivo a posteriori

Figura 3.9

Algoritmi per ricerca in ampiezza e ricerca a costo uniforme.

```

function RICERCA-IN-AMPIEZZA(problema) returns un nodo soluzione o fallimento
  nodo  $\leftarrow$  NODO(problema.STATOINIZIALE)
  if problema.È-OBIETTIVO(nodo.STATO) then return nodo
  frontiera  $\leftarrow$  una coda FIFO, con nodo come elemento iniziale
  raggiunti  $\leftarrow$  {problema.STATOINIZIALE}
  while not VUOTA?(frontiera) do
    nodo  $\leftarrow$  POP(frontiera)
    for each figlio in ESPANDI(problema, nodo) do
      s  $\leftarrow$  figlio.STATO
      if problema.È-OBIETTIVO(s) then return figlio
      if s non è in raggiunti then
        aggiungi s a raggiunti
        aggiungi figlio a frontiera
  return fallimento

function RICERCA-COSTO-UNIFORME(problema) returns un nodo soluzione o fallimento
  return RICERCA-BEST-FIRST(problema, COSTO-CAMMINO)

```



ricerca a costo uniforme

Tutti i nodi rimangono in memoria, perciò le complessità temporale e spaziale sono entrambe $O(b^d)$, quindi esponenziali, cosa che suscita preoccupazione. Come esempio tipico del mondo reale, consideriamo un problema con fattore di ramificazione $b = 10$, velocità di elaborazione di 1 milione di nodi al secondo e requisiti di memoria di 1 Kbyte/nodo. Una ricerca a profondità $d = 10$ richiederebbe meno di 3 ore, ma 10 terabyte di memoria. *Nella ricerca in ampiezza, i requisiti di memoria rappresentano un problema più importante rispetto al tempo di esecuzione.* Tuttavia, il tempo rimane comunque un fattore importante. A profondità $d = 14$, anche con memoria infinita, una ricerca richiederebbe 3,5 anni. In generale, *i problemi di ricerca con complessità esponenziale non possono essere risolti mediante ricerche non informate, se non nelle istanze più piccole.*

3.4.2 Algoritmo di Dijkstra o ricerca a costo uniforme

Quando le azioni hanno costi diversi, la scelta ovvia è quella di usare una ricerca best-first in cui la funzione di valutazione è il costo del cammino dalla radice al nodo corrente. Questa ricerca, che nella comunità dell'informatica teorica si chiama algoritmo di Dijkstra, nella comunità dell'IA si chiama **ricerca a costo uniforme**. L'idea è che mentre la ricerca in ampiezza si diffonde a ondate di profondità uniforme – prima profondità 1, poi 2 e così via – la ricerca a costo uniforme si diffonde a ondate di costo uniforme. L'algoritmo può essere implementato come chiamata di RICERCA-BEST-FIRST con COSTO-CAMMINO come funzione di valutazione (Figura 3.9).

Consideriamo ora la Figura 3.10, in cui il problema è andare da Sibiu a Bucarest. I nodi successori di Sibiu sono Rimnicu Vilcea e Fagaras, con rispettivi costi di 80 e 99. Il nodo di costo minimo è Rimnicu Vilcea, viene quindi espanso tale nodo, aggiungendo Pitesti con costo $80 + 97 = 177$. Ora il nodo di costo minimo è Fagaras, perciò viene espanso, aggiungendo Bucarest con costo $99 + 211 = 310$. Bucarest è l'obiettivo, ma l'algoritmo controlla se è stato raggiunto l'obiettivo soltanto quando espande un nodo, non quando lo genera, perciò non ha ancora scoperto che questo è un cammino che porta all'obiettivo.

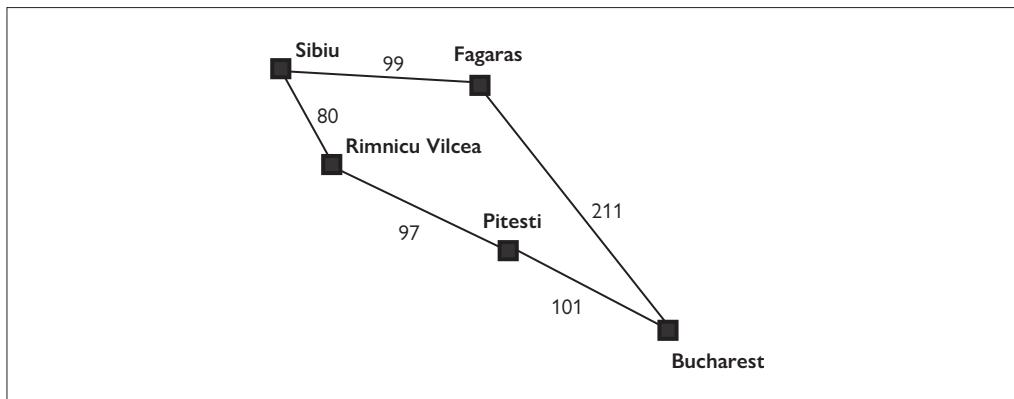


Figura 3.10
Una parte dello spazio degli stati per la Romania, selezionata per illustrare la ricerca a costo uniforme.

L'algoritmo continua, scegliendo Pitesti per la prossima espansione e aggiungendo un secondo cammino a Bucarest con costo $80 + 97 + 101 = 278$. Ha un costo inferiore, perciò sostituisce il precedente cammino in *raggiunti* ed è aggiunto a *frontiera*. Ora questo nodo ha il costo minimo, perciò l'algoritmo lo considera, determina che è un obiettivo e lo restituisce. Notate che, se avessimo controllato di aver trovato un obiettivo al momento di generare un nodo anziché al momento di espandere il nodo di costo minimo, avremmo restituito un cammino di costo più alto (quello che passa per Fagaras).

La complessità della ricerca a costo uniforme è caratterizzata in termini di C^* , il costo della soluzione ottima,⁸ ed ε , un limite inferiore imposto al costo di ogni azione, con $\varepsilon > 0$. Quindi, nel caso peggiore la complessità temporale e spaziale dell'algoritmo è $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$, che può essere molto maggiore di b^d . Ciò avviene perché la ricerca a costo uniforme può esplorare grandi alberi di azioni con costi bassi prima di esplorare cammini che comportano un costo elevato e forse azioni utili. Quando tutte le azioni hanno lo stesso costo, $b^{1+\lfloor C^*/\varepsilon \rfloor}$ è b^{d+1} e la ricerca a costo uniforme è simile alla ricerca in ampiezza.

La ricerca a costo uniforme è completa e ottima rispetto al costo, perché la prima soluzione che trova avrà un costo basso almeno quanto quello di ogni altro nodo sulla frontiera. Questa ricerca esamina sistematicamente tutti i cammini in ordine di costo crescente, quindi non entra mai in un cammino infinito (assumendo che tutti i costi delle azioni siano $> \varepsilon > 0$).

3.4.3 Ricerca in profondità e problema della memoria

La **ricerca in profondità** espande sempre per primo il nodo a *profondità maggiore* nella frontiera. Potrebbe essere implementata con una chiamata di RICERCA-BEST-FIRST in cui la funzione di valutazione f è l'opposto (negativo) della profondità. Tuttavia, generalmente viene implementata non come ricerca su grafo ma come ricerca ad albero che non mantiene una tabella degli stati raggiunti. Il progredire della ricerca è illustrato nella Figura 3.11: la ricerca raggiunge immediatamente il livello più profondo dell'albero, dove i nodi non hanno successori, quindi “torna indietro” al nodo più profondo che ha ancora successori non espansi. La ricerca in profondità non è ottima rispetto al costo, infatti restituisce sempre la prima soluzione che trova anche se non è la meno costosa.

ricerca in profondità

Per spazi degli stati finiti che sono alberi, la ricerca in profondità è efficiente e completa; per spazi degli stati aciclici potrebbe arrivare a espandere lo stesso stato più volte attraverso cammini diversi, ma (alla fine) esplorerà sistematicamente l'intero spazio.

⁸ Qui e in tutto il resto del libro l'asterisco in C^* indica un valore ottimo per C .

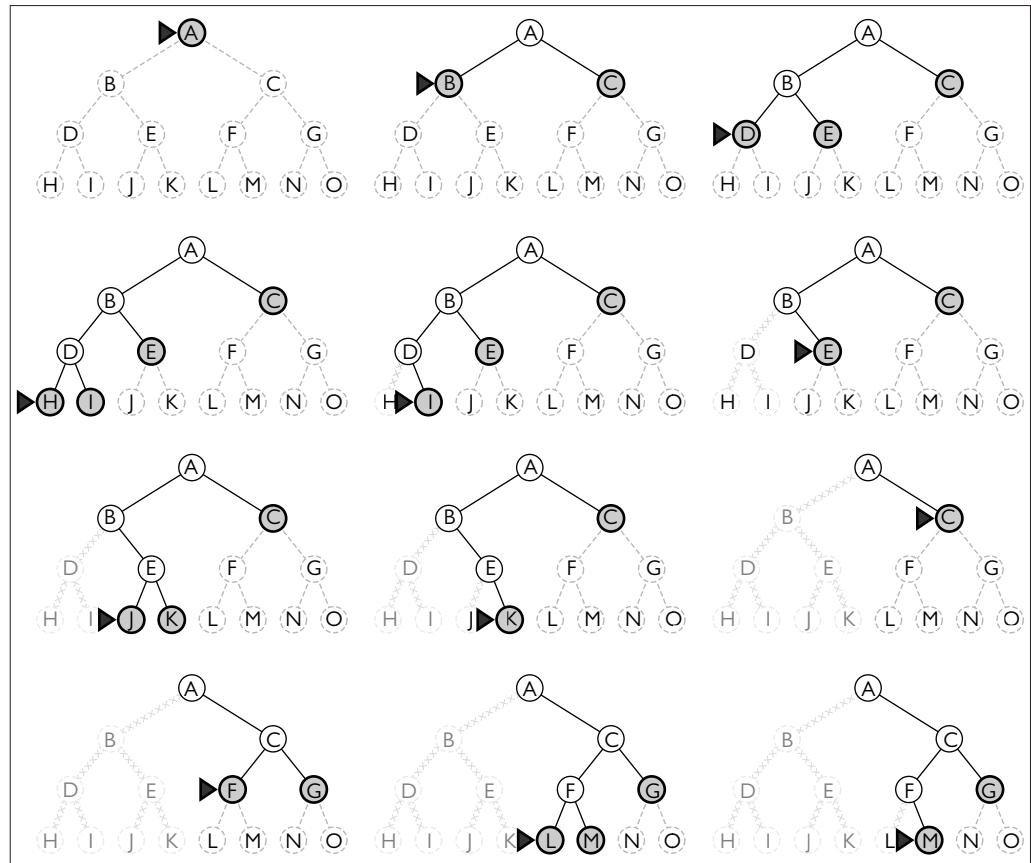


Figura 3.11 Una dozzina di passi (da sinistra a destra, dall'alto in basso) in una ricerca in profondità su un albero binario dallo stato di partenza A all'obiettivo M. La frontiera è evidenziata in grigio, con un triangolino che indica il prossimo nodo da espandere. I nodi già espansi in precedenza sono in bianco e i potenziali nodi futuri sono tratteggiati in chiaro. I nodi espansi senza discendenti nella frontiera (linee molto chiare) possono essere scartati.

In spazi degli stati ciclici, la ricerca in profondità può bloccarsi in un ciclo infinito, perciò alcune implementazioni controllano ciascun nodo nuovo per verificare la presenza di cicli. Infine, in spazi degli stati infiniti la ricerca in profondità non è sistematica: può entrare in un cammino infinito, anche se non vi sono cicli. Quindi, la ricerca in profondità è incompleta.

Con tutti questi aspetti problematici, perché si dovrebbe pensare di ricorrere alla ricerca in profondità anziché a quella in ampiezza o alla ricerca best-first? La risposta è che, per i problemi in cui è praticabile una ricerca ad albero, la ricerca in profondità ha esigenze di memoria molto più ridotte. Non viene mantenuta una tabella dei nodi raggiunti, e la frontiera è molto piccola: mentre nella ricerca in ampiezza la frontiera può essere vista come la superficie di una sfera in continua espansione, nella ricerca in profondità la frontiera è soltanto un raggio della sfera.

Per uno spazio degli stati finito e a forma di albero come quello della Figura 3.11, una ricerca ad albero in profondità richiede un tempo proporzionale al numero degli stati e ha una complessità di memoria solo $O(bm)$, dove b è il fattore di ramificazione e m è la massima profondità dell'albero. Alcuni problemi che richiederebbero exabyte di memoria con la ricerca in ampiezza possono essere affrontati con pochi kilobyte usando la ricerca in profondità. Grazie al fatto che richiede poca memoria, la ricerca (ad albero) in profondità è utilizzata.

zata ampiamente in molti campi dell'IA tra cui il soddisfacimento di vincoli (Capitolo 6), la soddisfabilità proposizionale (Capitolo 7) e la programmazione logica (Capitolo 9).

Esiste una variante della ricerca in profondità, la **ricerca con backtracking** (il termine significa letteralmente “ritornare sui propri passi”), che richiede ancora meno memoria; è descritta in dettaglio nel Capitolo 6. Nel backtracking, quando si espande un nodo non vengono generati tutti i successori, bensì uno solo; ogni nodo parzialmente espanso ricorda quale successore deve essere generato in seguito. In più, i successori sono generati *modificando* la descrizione dello stato corrente anziché allocando memoria per uno stato nuovo, e questo riduce i requisiti di memoria a una descrizione di stato e un cammino di $O(m)$ azioni, con un notevole risparmio rispetto agli $O(bm)$ stati da memorizzare nella ricerca in profondità. Con il backtracking, abbiamo anche la possibilità di mantenere una struttura dati efficiente per gli stati sul cammino corrente, che ci consente di controllare la presenza di un cammino ciclico in tempo $O(1)$ anziché $O(m)$. Affinché il backtracking funzioni, dobbiamo avere la possibilità di *annullare* ogni modifica quando “torniamo sui nostri passi”. Il backtracking è fondamentale per il successo in molti problemi caratterizzati da descrizioni di stato molto grandi, come il montaggio robotizzato.

ricerca con backtracking

3.4.4 Ricerca a profondità limitata e ad approfondimento iterativo

Per evitare che la ricerca in profondità si perda in un cammino infinito possiamo usare la **ricerca a profondità limitata**, in cui forniamo un limite di profondità, ℓ , e consideriamo tutti i nodi a profondità ℓ come se non avessero successori (Figura 3.12). La complessità temporale

ricerca a profondità limitata

```

function RICERCA-APPROFONDIMENTO-ITERATIVO(problema) returns un nodo soluzione o fallimento
  for profondità = 0 to  $\infty$  do
    risultato  $\leftarrow$  RICERCA-PROFONDITÀ-LIMITATA(problema, profondità)
    if risultato  $\neq$  soglia then return risultato

function RICERCA-PROFONDITÀ-LIMITATA(problema,  $\ell$ ) returns un nodo soluzione o fallimento o soglia
  frontiera  $\leftarrow$  una coda LIFO (stack) con NODO(problema.STATOINIZIALE) come elemento iniziale
  risultato  $\leftarrow$  fallimento
  while not VUOTA?(frontiera) do
    nodo  $\leftarrow$  POP(frontiera)
    if problema.È-OBIETTIVO(nodo.STATO) then return nodo
    if PROFONDITÀ(nodo)  $>$   $\ell$  then
      risultato  $\leftarrow$  soglia
    else if not È-CICLO(nodo) do
      for each figlio in ESPANDI(problema, nodo) do
        aggiungi figlio a frontiera
    return risultato
  
```

Figura 3.12 Ricerca ad approfondimento iterativo e ricerca (ad albero) a profondità limitata. L'approfondimento iterativo esegue ripetutamente una ricerca a profondità limitata aumentando via via il limite. Restituisce uno fra tre tipi di valori: un nodo soluzione; oppure *fallimento* quando ha esaurito tutti i nodi dimostrando che non esiste alcuna soluzione ad alcuna profondità; oppure *soglia* a indicare che potrebbe esistere una soluzione a un livello di profondità superiore a ℓ . Questo è un algoritmo di ricerca ad albero che non tiene traccia degli stati raggiunti e quindi utilizza molta meno memoria rispetto alla ricerca best-first, ma corre il rischio di visitare lo stesso stato più volte su cammini diversi. Inoltre, se il controllo È-CICLO non controlla tutti i cammini ciclici, l'algoritmo potrebbe entrare in un loop.

è $O(b^\ell)$ e quella spaziale è $O(b\ell)$. Sfortunatamente, se si sceglie male ℓ l’algoritmo non riuscirà a trovare la soluzione, per cui è ancora incompleto.

Poiché la ricerca in profondità è una ricerca ad albero, non possiamo evitare che perda tempo su cammini ridondanti in generale, ma possiamo eliminare i cammini ciclici al costo di un po’ di tempo di calcolo in più. Se esaminiamo soltanto pochi collegamenti risalendo nella catena di parentela, riusciamo a catturare la maggior parte dei cicli; quelli più lunghi sono rimossi grazie al limite sulla profondità.

diametro

A volte si può scegliere un buon limite di profondità basandosi sulla conoscenza del problema. Per esempio, nella nostra mappa della Romania ci sono 20 città, quindi $\ell = 19$ è un limite valido. Ma se studiassimo la mappa con attenzione scopriremmo che ogni città può essere raggiunta da ogni altra in al più 9 azioni. Questo numero, noto come **diametro** dello spazio degli stati, ci fornisce un limite di profondità preferibile al precedente che porta a una ricerca a profondità limitata più efficiente. Tuttavia, nella maggior parte dei problemi non riusciremo a sapere quale sia un buon limite di profondità finché non avremo risolto il problema.

ricerca ad approfondimento iterativo

La **ricerca ad approfondimento iterativo** risolve il problema di trovare un buon valore di ℓ provando tutti i valori: prima 0, poi 1, poi 2, e così via fino a quando si trova una soluzione, oppure la ricerca a profondità limitata restituisce il valore *fallimento* anziché il valore *soglia*. L’algoritmo è riportato nella Figura 3.12. L’approfondimento iterativo riunisce molti vantaggi offerti dalla ricerca in profondità e da quella in ampiezza. Come nella ricerca in profondità, i requisiti di memoria sono modesti: $O(bd)$ quando esiste una soluzione, $O(bm)$ su spazi degli stati finiti senza soluzione. Come la ricerca in ampiezza, la ricerca ad approfondimento operativo è ottima per problemi in cui tutte le azioni hanno lo stesso costo, ed è completa su spazi degli stati aciclici finiti, o su qualsiasi spazio degli stati finito quando controlliamo i nodi per la presenza di cicli risalendo l’intero cammino.

La complessità temporale è $O(b^d)$ quando esiste una soluzione, $O(b^m)$ quando non esiste. Ogni iterazione della ricerca ad approfondimento iterativo genera un nuovo livello, come la ricerca in ampiezza, ma mentre quest’ultima lo fa memorizzando tutti i nodi in memoria, la ricerca ad approfondimento iterativo lo fa ricostruendo tutti i livelli precedenti, risparmiando memoria in cambio di un tempo maggiore. La Figura 3.13 mostra quattro iterazioni di una ricerca ad approfondimento iterativo su un albero di ricerca binario, dove la soluzione è trovata alla quarta iterazione.

La ricerca ad approfondimento iterativo potrebbe sembrare uno spreco, dato che gli stati vicini all’inizio della ricerca vengono rigenerati più volte. Tuttavia, in molti spazi degli stati la maggior parte dei nodi si trova al livello più basso, perciò la ripetizione dei livelli superiori non ha un grande impatto. In una ricerca ad approfondimento iterativo, i nodi al livello più basso (a profondità d) sono generati una sola volta, quelli al livello immediatamente superiore due volte, e così via fino ai figli diretti del nodo radice, che sono generati d volte. Quindi, nel caso peggiore il numero totale dei nodi generati è:

$$N(RAI) = (d)b^1 + (d-1)b^2 + \cdots + (1)b^d,$$

il che ci dà una complessità temporale $O(b^d)$, asintoticamente equivalente alla ricerca in ampiezza. Per esempio, per $b = 10$ e $d = 5$ i risultati sono:

$$N(RAI) = 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450$$

$$N(RIA) = 10 + 100 + 1.000 + 10.000 + 100.000 = 111.110.$$



Se siete realmente preoccupati in merito alla ripetizione, potete utilizzare un approccio ibrido che esegua la ricerca in ampiezza finché non si è esaurita quasi tutta la memoria disponibile, e poi un approfondimento iterativo da tutti i nodi della frontiera. *In generale, la ricerca ad approfondimento iterativo è il metodo preferito di ricerca non informata quando lo spazio degli stati in cui cercare è troppo grande per essere mantenuto in memoria e la profondità della soluzione non è nota.*

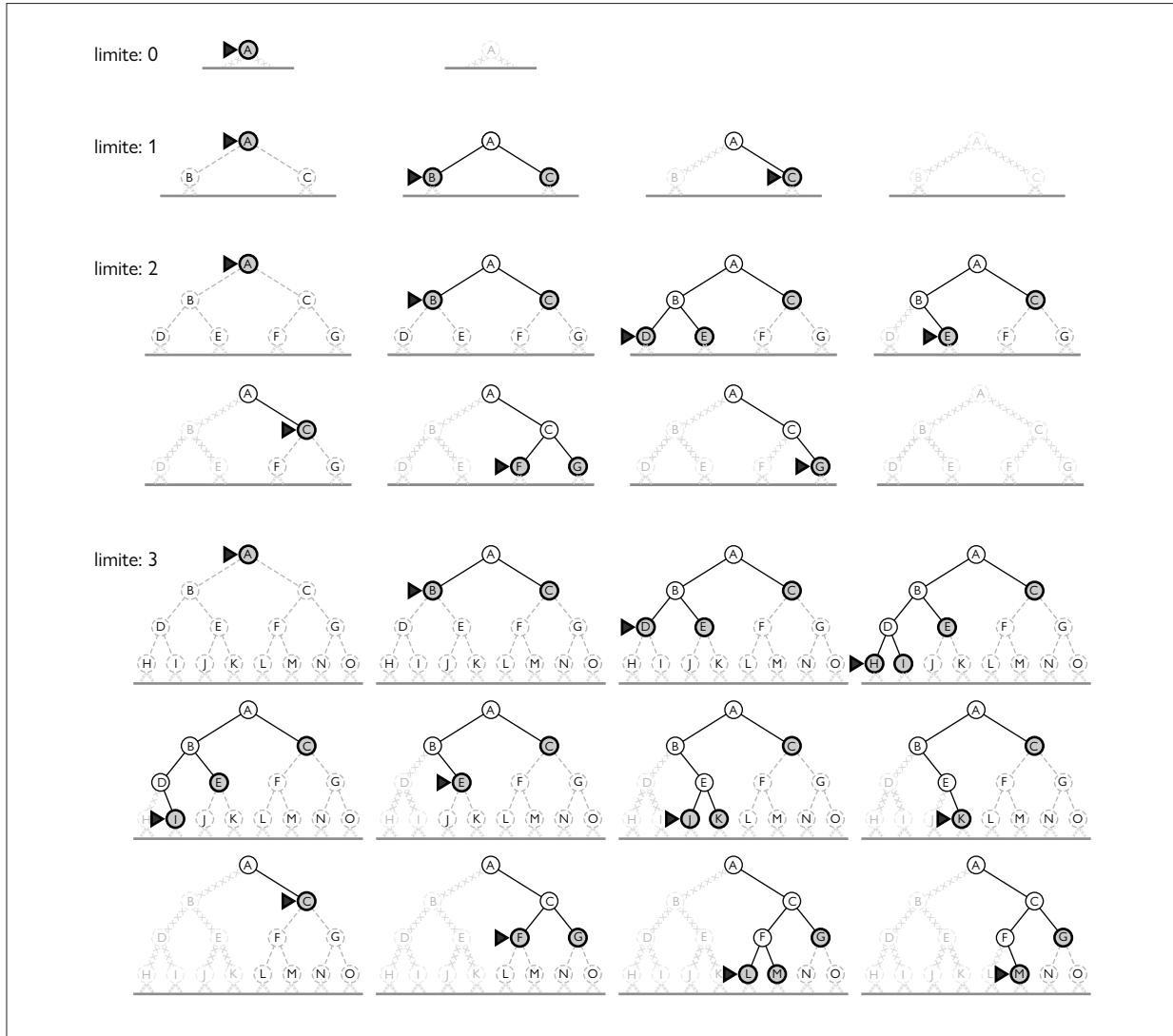


Figura 3.13 Quattro iterazioni di ricerca ad approfondimento iterativo per l’obiettivo M su un albero binario, con il limite di profondità che varia da 0 a 3. Notate che i nodi interni rispetto alla frontiera formano un cammino singolo. Il triangolino segnala il prossimo nodo da espandere; i nodi in grigio con contorno scuro sono sulla frontiera, per quelli molto chiari è dimostrato che non possono far parte della soluzione con il limite di profondità corrente.

3.4.5 Ricerca bidirezionale

Gli algoritmi che abbiamo trattato finora partono da uno stato iniziale e possono raggiungere ognuno dei possibili stati obiettivo. Un approccio alternativo è quello della **ricerca bidirezionale**, che ricerca procedendo simultaneamente in avanti a partire dallo stato iniziale e all’indietro a partire dallo stato obiettivo (o dagli stati obiettivi, se sono più di uno) nella speranza che le due ricerche si incontrino. L’idea di base è che $b^{d/2} + b^{d/2}$ è molto minore di b^d (per esempio, 50.000 volte minore quando $b = d = 10$).

Affinché questa strategia funzioni, è necessario tenere traccia di due frontiere e due tabelle di stati raggiunti, e occorre essere in grado di ragionare all’indietro: se lo stato s' è un

ricerca
bidirezionale

successore di s nella direzione in avanti, allora dobbiamo sapere che s è un successore di s' nella direzione all'indietro. Abbiamo una soluzione quando le due frontiere collidono.⁹

Esistono molte versioni differenti della ricerca bidirezionale, come avviene del resto per gli algoritmi di ricerca unidirezionale. In questo paragrafo descriviamo la ricerca bidirezionale best-first. Anche se ci sono due frontiere separate, il prossimo nodo da espandere è sempre quello con un valore minimo della funzione di valutazione. Quando la funzione di valutazione è il costo di cammino otteniamo una ricerca bidirezionale a costo uniforme, e se il costo del cammino ottimale è C^* , nessun nodo con costo $> C^*/2$ verrà espanso, cosa che può velocizzare notevolmente l'esecuzione.

L'algoritmo di ricerca bidirezionale best-first generale è mostrato nella Figura 3.14. Passiamo all'algoritmo due versioni del problema e la funzione di valutazione, una nella direzione in avanti (indice F) e una all'indietro (indice B). Quando la funzione di valutazione è il costo di cammino sappiamo che la prima soluzione trovata sarà una soluzione ottima, ma con funzioni di valutazione diverse non è necessariamente così, quindi teniamo traccia della migliore soluzione trovata finora, che potrebbe essere aggiornata più volte prima che il test TERMINATO dimostri che non può esistere un'altra soluzione migliore.

3.4.6 Confronto tra le strategie di ricerca non informata

La Figura 3.15 mette a confronto vari algoritmi di ricerca non informata secondo i quattro criteri di valutazione definiti nel Paragrafo 3.3.4. Tale confronto riguarda le versioni di ricerca ad albero senza controllo di stati ripetuti. Per le ricerche su grafo che effettuano tale controllo, le differenze principali sono che la ricerca in profondità è completa per spazi degli stati finiti e che le complessità spaziale e temporale sono limitate dalla dimensione dello spazio degli stati (il numero di vertici e archi, $|V| + |E|$).

3.5 Strategie di ricerca informata o euristica

ricerca informata

Questo paragrafo mostra come una strategia di **ricerca informata**, che sfrutta conoscenza specifica del dominio applicativo per fornire suggerimenti su dove si potrebbe trovare l'obiettivo, possa trovare soluzioni in modo più efficiente di una strategia non informata. I suggerimenti hanno la forma di una **funzione euristica** denotata con $h(n)$:¹⁰

$h(n) = \text{costo stimato del cammino meno costoso dallo stato del nodo } n \text{ a uno stato obiettivo.}$

Per esempio, in problemi di ricerca dell'itinerario possiamo stimare la distanza dallo stato corrente a uno stato obiettivo calcolando la distanza in linea d'aria tra due punti sulla mappa. Le euristiche e come si possono ricavare sono trattate in maggiore dettaglio nel Paragrafo 3.6.

3.5.1 Ricerca best-first greedy o “golosa”

ricerca best-first greedy

La **ricerca best-first greedy** (il termine *greedy* viene talvolta tradotto in italiano come “golosa” o “avidità”) è una forma di ricerca best-first che espande prima il nodo con il valore più

⁹ Nella nostra implementazione, la struttura dati *raggiunti* supporta una query che determina se un dato stato ne fa parte, e la struttura dati *frontiera* (una coda con priorità) no, perciò controlliamo l'esistenza di una collisione usando *raggiunti*; tuttavia, concettualmente ci stiamo chiedendo se le due frontiere si sono incontrate. L'implementazione può essere estesa per gestire più stati obiettivo caricando il nodo per ciascuno di essi nella frontiera “all'indietro” e nella tabella *raggiunti* corrispondente.

¹⁰ Potrebbe sembrare strano che la funzione euristica operi su un nodo, quando tutto ciò che le serve è lo stato del nodo, ma è consuetudine utilizzare $h(n)$ anziché $h(s)$ per coerenza con la funzione di valutazione $f(n)$ e il costo di cammino $g(n)$.

```

function RICERCA-BEST-FIRST-BIDIREZIONALE(problemaF, fF, problemaB, fB) returns un nodo soluzione o fallimento
  nodoF  $\leftarrow$  NODO(problemaF.STATOINIZIALE) // Nodo per uno stato iniziale
  nodoB  $\leftarrow$  NODO(problemaB.STATOINIZIALE) // Nodo per uno stato obiettivo
  frontieraF  $\leftarrow$  coda con priorità ordinata in base a fF, con nodoF come elemento iniziale
  frontieraB  $\leftarrow$  coda con priorità ordinata in base a fB, con nodoB come elemento iniziale
  raggiuntiF  $\leftarrow$  tabella di lookup, con un elemento con chiave nodoF.STATO e valore nodoF
  raggiuntiB  $\leftarrow$  tabella di lookup, con un elemento con chiave nodoB.STATO e valore nodoB
  soluzione  $\leftarrow$  fallimento
  while not TERMINATO(soluzione, frontieraF, frontieraB) do
    if fF(TOP(frontieraF)) < fB(TOP(frontieraB)) then
      soluzione  $\leftarrow$  PROCEDI(F, problemaF, frontieraF, raggiuntiF, raggiuntiB, soluzione)
    else soluzione  $\leftarrow$  PROCEDI(B, problemaB, frontieraB, raggiuntiB, raggiuntiF, soluzione)
  return soluzione

function PROCEDI(dir, problema, frontiera, raggiunti, raggiunti2, soluzione) returns una soluzione
  // Espande nodo su frontiera; controlla rispetto all'altra frontiera in raggiunti2.
  // La variabile "dir" è la direzione: F per avanti o B per indietro.
  nodo  $\leftarrow$  POP(frontiera)
  for each figlio in ESPANDI(problema, nodo) do
    s  $\leftarrow$  figlio.STATO
    if s not in raggiunti or COSTO-CAMMINO(figlio) < COSTO-CAMMINO(raggiunti[s]) then
      raggiunti[s]  $\leftarrow$  figlio
      aggiungi figlio a frontiera
      if s è in raggiunti2 then
        soluzione2  $\leftarrow$  UNISCI-NODI(dir, figlio, raggiunti2[s]))
        if COSTO-CAMMINO(soluzione2) < COSTO-CAMMINO(soluzione) then
          soluzione  $\leftarrow$  soluzione2
  return soluzione

```

Figura 3.14 La ricerca best-first bidirezionale mantiene due frontiere e due tabelle di stati raggiunti. Quando un cammino di una frontiera raggiunge uno stato che è stato raggiunto anche nell'altra metà della ricerca, i due cammini sono congiunti (con la funzione UNISCI-NODI) a formare una soluzione. La prima soluzione ottenuta non è necessariamente la migliore; la funzione TERMINATO determina quando fermare la ricerca di nuove soluzioni.

criterio	in ampiezza	a costo uniforme	in profondità	a profondità limitata	ad approfondimento iterativo	bidirezionale (se applicabile)
completa?	Sì ¹	Sì ^{1,2}	No	No	Sì ¹	Sì ^{1,4}
ottima rispetto al costo?	Sì ³	Sì	No	No	Sì ³	Sì ^{3,4}
tempo	$O(b^d)$	$O(b^{1+\lfloor C/\varepsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
spazio	$O(b^d)$	$O(b^{1+\lfloor C/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

Figura 3.15 Valutazione di algoritmi di ricerca. *b* è il fattore di ramificazione; *m* la profondità massima dell'albero di ricerca; *d* la profondità della soluzione più vicina alla radice, o è uguale a *m* quando non vi è soluzione; *ℓ* è il limite alla profondità. Gli apici sono da interpretare come segue: ¹ completa se *b* è finito, e lo spazio degli stati ha una soluzione o è finito; ² completa se tutti i costi di azione sono $\geq \varepsilon > 0$; ³ ottima rispetto al costo se i costi delle azioni sono tutti identici; ⁴ se in entrambe le direzioni si usa una ricerca in ampiezza o una ricerca a costo uniforme.

Figura 3.16

Valori di h_{DLA} – distanze in linea d'aria verso Bucarest.

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

distanza in linea d'aria

basso di $h(n)$, cioè quello che appare più vicino all'obiettivo, sulla base del fatto che è probabile che questo porti rapidamente a una soluzione. Perciò la funzione di valutazione è $f(n) = h(n)$.

Vediamo come funziona con il nostro problema dell'itinerario in Romania usando l'euristica della **distanza in linea d'aria**, che indicheremo con h_{DLA} . Se l'obiettivo è Bucarest, dovremo conoscere tutte le distanze in linea d'aria verso quella città, che sono elencate nella Figura 3.16. Per esempio, $h_{DLA}(Arad) = 366$. Notate che i valori di h_{DLA} non possono essere calcolati direttamente dalla descrizione del problema (cioè dalle funzioni AZIONI e RISULTATO). Inoltre, serve una certa esperienza per sapere che h_{DLA} è effettivamente correlata alle distanze su strada e rappresenta quindi un'euristica utile.

La Figura 3.17 mostra i progressi di una ricerca best-first greedy che usa h_{DLA} per trovare un cammino da Arad a Bucarest. Il primo nodo espanso da Arad sarà Sibiu, perché l'euristica afferma che è più vicino a Bucarest di Zerind e Timisoara. Il nodo successivo sarà Fagaras, perché ora è il più vicino secondo l'euristica. Fagaras a sua volta genererà Bucarest, che è l'obiettivo. In questo particolare problema, la ricerca best-first greedy che usa h_{DLA} trova una soluzione senza mai neppure espandere un nodo che non sia sul cammino della soluzione. Tuttavia, la soluzione trovata non ha un costo ottimo, infatti il cammino che passa da Sibiu e Fagaras è di 32 miglia più lungo di quello che attraversa Rimnicu Vilcea e Pitesti. Ecco perché l'algoritmo è chiamato “goloso”: a ogni passo cerca sempre di arrivare il più possibile vicino a un obiettivo; tuttavia, questa golosità può portare a risultati peggiori di quelli ottenibili procedendo con maggiore cautela.

La ricerca best-first greedy su grafo è completa negli spazi degli stati finiti, ma non in quelli infiniti. Nel caso peggiore la complessità temporale e spaziale è $O(|V|)$, ma con una buona funzione euristica è possibile ridurla considerevolmente, arrivando in certi problemi a $O(bm)$.

3.5.2 Ricerca A*

ricerca A*

La forma più diffusa di algoritmo di ricerca informata è la **ricerca A*** (si dice “ricerca A-stella” o “ricerca A-star”), una ricerca best-first che utilizza la funzione di valutazione:

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

dove $g(n)$ è il costo del cammino dal nodo iniziale al nodo n e $h(n)$ rappresenta il costo *stimato* del cammino più breve da n a uno stato obiettivo, per cui abbiamo:

$$f(n) = \text{costo stimato del cammino migliore che continua da } n \text{ fino a un obiettivo.}$$

Nella Figura 3.18 è mostrato il progredire di una ricerca A* con l'obiettivo di raggiungere Bucarest. I valori di g sono calcolati dai costi di azione della Figura 3.1, e i valori di h_{DLA}

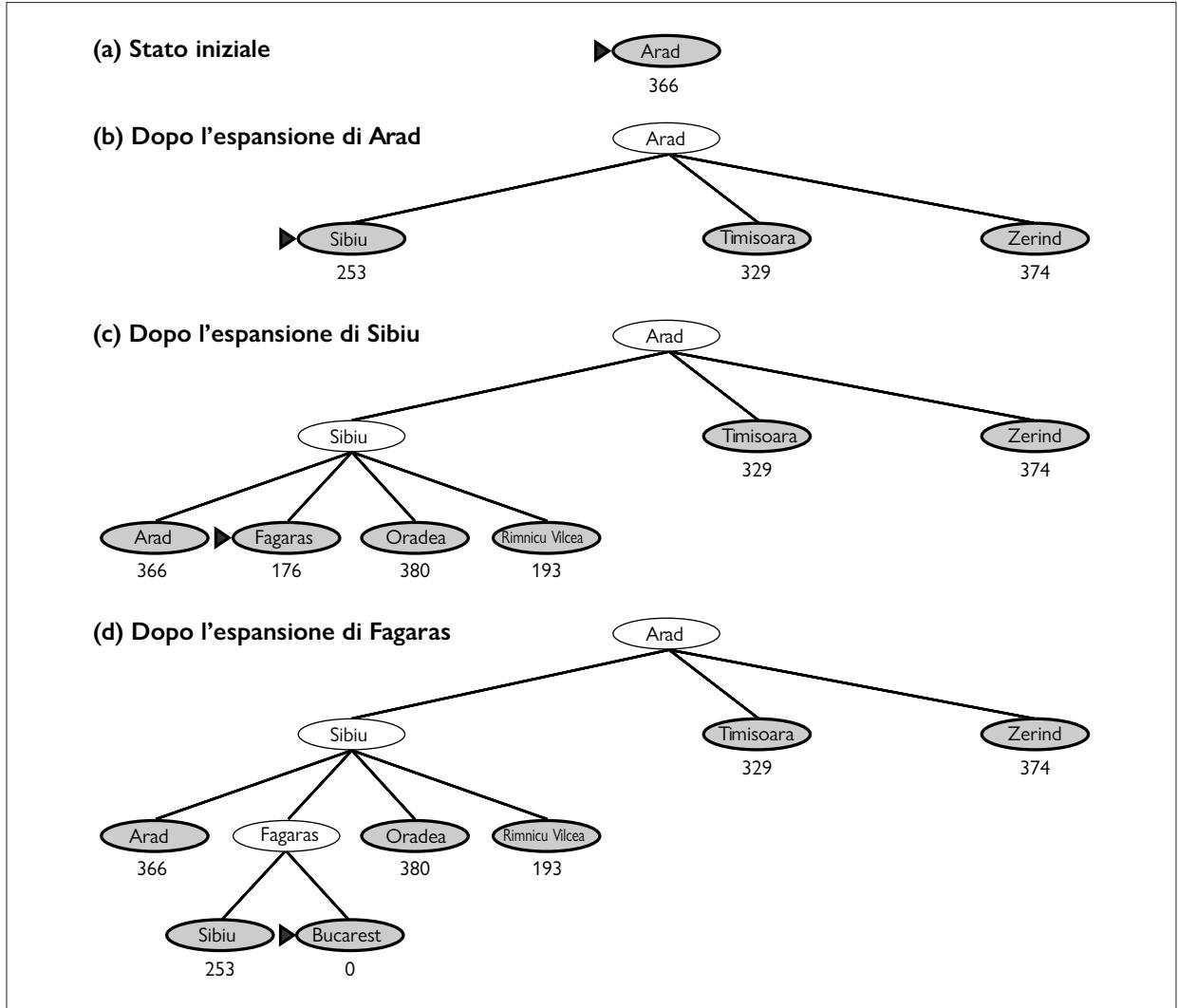


Figura 3.17 Passi di una ricerca (ad albero) best-first greedy di un itinerario verso Bucarest che usa l'euristica della distanza in linea d'aria h_{DLA} . I nodi sono etichettati con i loro valori h .

sono forniti nella Figura 3.16. Notate che Bucarest appare prima sulla frontiera al passo (e), ma non è selezionata per l'espansione (e quindi non individuata come soluzione) perché con $f = 450$ non è il nodo di costo minimo sulla frontiera – che, invece, è Pitesti con $f = 417$. In altre parole, questo significa che *potrebbe* esistere una soluzione attraverso Pitesti con costo di 417, per cui l'algoritmo non si ferma a una soluzione di costo 450. Al passo (f), un diverso cammino verso Bucarest è il nodo di costo minimo, con $f = 418$, per cui viene selezionato e individuato come soluzione ottima.

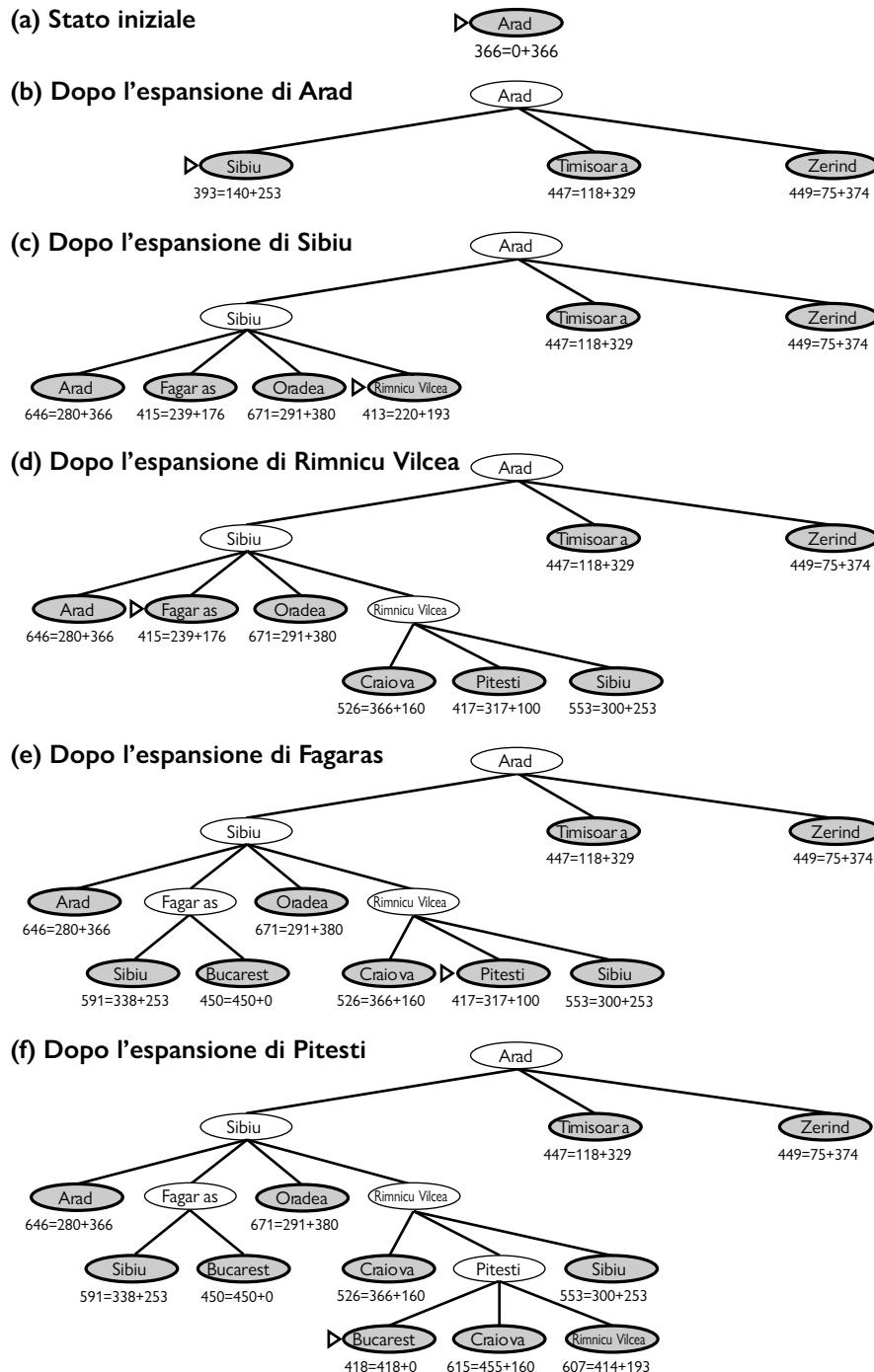
La ricerca A* è completa.¹¹ Il fatto che sia o meno ottimale rispetto al costo dipende da alcune proprietà dell'euristica. Una proprietà fondamentale è l'**ammissibilità**: un'**euristica ammissibile**

euristica ammissibile

¹¹ Anche qui si assume che tutti i costi di azione siano $> \varepsilon > 0$ e che lo spazio degli stati abbia una soluzione o sia finito.

Figura 3.18

Passi di una ricerca A* di un itinerario verso Bucarest. I nodi sono etichettati con i valori $f = g + h$. I valori h sono distanze in linea d'aria verso Bucarest, prese dalla Figura 3.16.



ammissibile è tale se *non sovrastima mai* il costo per raggiungere un obiettivo (un'euristica ammissibile è, quindi, ottimista). Con un'euristica ammissibile la ricerca A* è ottima rispetto al costo, cosa che possiamo dimostrare per assurdo. Supponiamo che il cammino ottimo abbia costo C^* , ma l'algoritmo restituisca un cammino di costo maggiore: $C > C^*$. Allora deve

esistere un nodo n che si trova sul cammino ottimo e non è espanso (perché se tutti i nodi sul cammino ottimo fossero stati espansi, allora sarebbe stata restituita la soluzione ottima). Allora, indicando con $g^*(n)$ il costo del cammino ottimo dall'inizio a n e con $h^*(n)$ il costo del cammino ottimo da n fino al più vicino obiettivo, abbiamo:

$$\begin{array}{ll} f(n) > C^* & \text{(altrimenti } n \text{ sarebbe stato espanso)} \\ f(n) = g(n) + h(n) & \text{(per definizione)} \\ f(n) = g^*(n) + h(n) & \text{(dato che } n \text{ si trova su un cammino ottimo)} \\ f(n) \leq g^*(n) + h^*(n) & \text{(per l'ammissibilità, } h(n) \leq h^*(n)) \\ f(n) \leq C^* & \text{(per definizione, } C^* = g^*(n) + h^*(n)) \end{array}$$

La prima e l'ultima riga si contraddicono, perciò l'ipotesi che l'algoritmo possa restituire un cammino subottimo deve essere errata, quindi la ricerca A^* restituisce soltanto cammini ottimi rispetto al costo.

Una proprietà leggermente più forte è la **consistenza**. Un'euristica $h(n)$ è consistente se, per ogni nodo n e ogni successore n' di n generato da un'azione a , abbiamo:

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

Questa è una forma della **disuguaglianza triangolare**, per cui ogni lato di un triangolo non può essere più lungo della somma degli altri due (Figura 3.19). Un esempio di euristica consistente è la distanza in linea d'aria h_{DLA} che abbiamo usato nell'arrivare a Bucarest.

consistenza

disuguaglianza triangolare

Ogni euristica consistente è ammissibile (ma non vale il vice versa), perciò A^* con un'euristica consistente è ottima rispetto al costo. Inoltre, con un'euristica consistente, la prima volta che raggiungiamo uno stato sarà su un cammino ottimo, perciò non dovremo mai aggiungere di nuovo uno stato alla frontiera, né dovremo mai modificare un elemento in *raggiunti*. Con un'euristica inconsistente, invece, potremmo ritrovarci con più cammini che raggiungono lo stesso stato, e se ogni cammino nuovo ha un costo inferiore al precedente, finiremo con l'avere più nodi corrispondenti a quello stato sulla frontiera, con un aggravio di costo temporale e spaziale. Per questo motivo alcune implementazioni di A^* verificano di inserire uno stato nella frontiera una sola volta, e se viene trovato un cammino migliore che porta a quello stato, tutti i suoi successori vengono aggiornati (questo richiede che i nodi abbiano puntatori figli oltre che puntatori padre). Queste complicazioni hanno portato molti implementatori a evitare di usare euristiche inconsistenti, ma Felner *et al.* (2011) sostengono che gli effetti peggiori si presentano di rado nella pratica, per cui non è il caso di temere di usare euristiche inconsistenti.

Con un'euristica inammissibile, A^* può essere ottima rispetto al costo oppure no. Due casi in cui lo è sono i seguenti: primo, se vi è anche un solo cammino ottimo rispetto al costo lungo cui $h(n)$ è ammissibile per tutti i nodi n sul cammino, allora tale cammino verrà trovato a prescindere da quanto affermi l'euristica per gli stati al di fuori di esso. Secondo, se la soluzione ottima ha costo C^* e la seconda migliore ha costo C_2 , e se $h(n)$ sovrastima alcuni costi, ma mai più di $C_2 - C^*$, allora A^* restituisce sempre soluzioni ottime rispetto al costo.

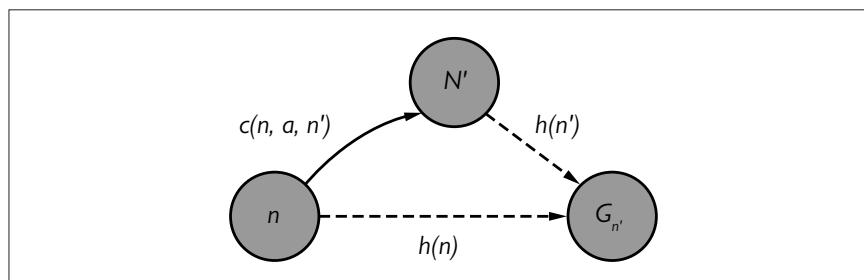


Figura 3.19 Diseguaglianza triangolare: se l'euristica h è **consistente**, allora il singolo numero $h(n)$ sarà minore della somma del costo $c(n,a,n')$ dell'azione per andare da n a n' e della stima euristica $h(n')$.

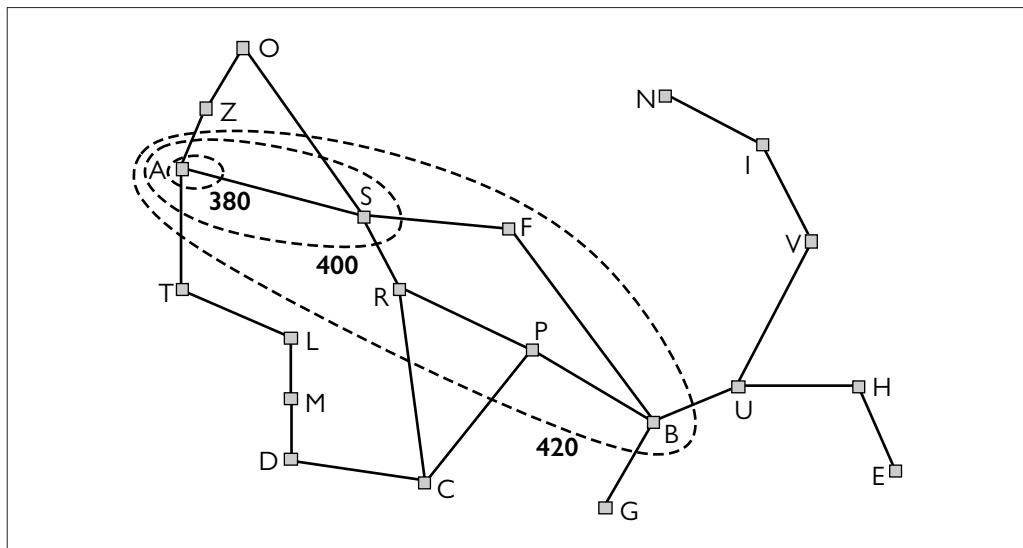


Figura 3.20 Mappa della Romania che indica i confini per $f = 380$, $f = 400$ e $f = 420$ con Arad come stato iniziale. I nodi all'interno di un dato confine hanno tutti un costo $f = g + h$ minore o uguale al valore del confine. Per motivi di spazio, in questa figura le città sono indicate con la lettera iniziale del loro nome (cfr. Figura 3.1 per i nomi completi).

3.5.3 Confini di ricerca

Un modo utile per visualizzare una ricerca consiste nel disegnare i confini nello spazio degli stati, come si fa in una mappa topografica. Un esempio è quello della Figura 3.20: all'interno del confine etichettato con 400 tutti i nodi hanno $f(n) = g(n) + h(n) \leq 400$, e così via. Allora, poiché A^* espande il nodo della frontiera con costo f minimo, possiamo vedere che una ricerca A^* si espanderà a partire dal nodo iniziale aggiungendo nodi in bande concentriche di f -costo crescente.

Anche nella ricerca a costo uniforme abbiamo i confini, ma di costo g , non $g + h$. I confini nella ricerca a costo uniforme saranno “circolari” attorno allo stato iniziale e si diffonderanno uniformemente in tutte le direzioni, senza preferenza verso l'obiettivo. Con la ricerca A^* e una buona euristica, le bande $g + h$ si allungheranno verso uno stato obiettivo (Figura 3.20) e diventeranno sempre più strettamente focalizzate attorno a un cammino ottimo.

monotonò

Dovrebbe essere chiaro che, quando si estende un cammino, i costi g sono **monotoni**: il costo di cammino aumenta sempre mentre lo si percorre, poiché i costi di azione sono sempre positivi.¹² Abbiamo quindi linee di confine concentriche che non si attraversano mai le une con le altre, e se scegliamo di disegnare le linee con sufficiente dettaglio, possiamo porre una linea tra due nodi qualsiasi su qualsiasi cammino.

Non è invece chiaro se il costo $f = g + h$ è monotono crescente. Quando si estende un cammino da n a n' , il costo da $g(n) + h(n)$ diventa $g(n) + c(n, a, n') + h(n')$. Eliminando il termine $g(n)$ vediamo che il costo del cammino sarà monotono crescente se e solo se $h(n) \leq c(n, a, n') + h(n')$; in altre parole, se e solo se l'euristica è consistente.¹³ Notate però che un cammino potrebbe avere diversi nodi di fila con lo stesso valore $g(n) + h(n)$; ciò accade ogni volta che

¹² Tecnicamente indichiamo come “strettamente monotoni” i costi che aumentano sempre e “monotoni” i costi che non diminuiscono mai, ma possono rimanere invariati.

¹³ In effetti il termine “euristica monotonà” è sinonimo di “euristica consistente”. I due concetti sono stati sviluppati in modo indipendente e in seguito si è dimostrata la loro equivalenza (Pearl, 1984).

la diminuzione di h è esattamente pari al costo di azione appena sostenuto (per esempio, in un problema su griglia, quando n è nella stessa riga dell'obiettivo e si fa un passo verso l'obiettivo, g aumenta di 1 e h diminuisce di 1). Se C^* è il costo del cammino della soluzione ottima, possiamo dire quanto segue:

- A^* espande tutti i nodi che possono essere raggiunti dallo stato iniziale su un cammino in cui per ogni nodo si ha $f(n) < C^*$. Diciamo che questi sono **nodi certamente espansi**.
- A^* potrebbe poi espandere alcuni dei nodi proprio sul “confine obiettivo” (dove $f(n) = C^*$) prima di selezionare un nodo obiettivo.
- A^* non espande alcun nodo con $f(n) > C^*$.

nodo certamente espanso

Diciamo che A^* con un'euristica consistente è **ottimamente efficiente** nel senso che qualsiasi algoritmo che estende cammini di ricerca a partire dallo stato iniziale e usa la stessa euristica deve espandere tutti i nodi che sono certamente espansi da A^* (perché uno qualunque di essi potrebbe essere parte di una soluzione ottima). Tra i nodi con $f(n) = C^*$, un algoritmo potrebbe avere fortuna e trovare quello ottimo prima, mentre un altro potrebbe essere sfortunato; non consideriamo questa differenza nella definizione di efficienza ottimale.

ottimamente efficiente

A^* è efficiente perché esegue la **potatura** dall'albero di ricerca dei nodi non necessari per trovare una soluzione ottima. Nella Figura 3.18(b) vediamo che Timisoara ha $f = 447$ e Zerind ha $f = 449$. Anche se sono entrambi figli della radice e sarebbero tra i primi nodi espansi da una ricerca a costo uniforme o in ampiezza, questi nodi non sono mai espansi da una ricerca A^* perché la soluzione con $f = 418$ viene trovata prima. Il concetto di potatura – scaricare alcune possibilità senza doverle nemmeno esaminare – è importante per molti campi dell'IA.

potatura

Il fatto che la ricerca A^* sia completa, ottima rispetto al costo e ottimamente efficiente tra tutti gli algoritmi del genere è assai soddisfacente, ma purtroppo non significa che A^* sia la risposta a tutte le nostre esigenze di ricerca. L'inghippo è che per molti problemi il numero di nodi espansi può aumentare esponenzialmente con la lunghezza della soluzione. Per esempio, consideriamo una versione del mondo dell'aspirapolvere con un apparecchio super potente in grado di ripulire qualsiasi cella al costo di 1 unità senza nemmeno doverla visitare; in tale scenario, le celle possono essere ripulite in qualsiasi ordine. Con N celle inizialmente sporche, ci sono 2^N stati in cui un sottoinsieme delle celle è stato ripulito; tutti questi stati si trovano sul cammino di una soluzione ottima, quindi soddisfano $f(n) < C^*$, perciò tutti sarebbero visitati da A^* .

3.5.4 Ricerca soddisfacente: euristiche inammissibili e ricerca A^* pesata

La ricerca A^* ha molte buone qualità, ma espande una grande quantità di nodi. È possibile esplorare un numero minore di nodi (con risparmio di tempo e di spazio) se si è disponibili ad accettare soluzioni subottime ma “sufficientemente buone”, che chiamiamo soluzioni **soddisfacenti** (*satisficing*). Se consentiamo alla ricerca A^* di usare un'**euristica inammissibile** – che potrebbe sovrastimare – rischiamo di non trovare la soluzione ottima, ma l'euristica potrebbe essere più accurata, riducendo così il numero di nodi espansi. Per esempio, gli ingegneri civili conoscono il concetto di **indice di deviazione**, un coefficiente moltiplicatore applicato alla distanza in linea d'aria per tenere conto dei cambi di curvatura delle strade. Un indice di deviazione di 1,3 significa che, se due città sono distanti 10 chilometri in linea d'aria, una buona stima del miglior percorso stradale per andare da una all'altra è 13 chilometri. Per la maggior parte delle località l'indice di deviazione è compreso tra 1,2 e 1,6.

euristica inammissibile

indice di deviazione

Possiamo applicare questa idea a qualsiasi problema, non solo a quelli che trattano di strade, con un approccio chiamato **ricerca A^* pesata** in cui diamo un peso maggiore al valore dell'euristica, con la funzione di valutazione $f(n) = g(n) + W \times h(n)$, per $W > 1$.

ricerca A^* pesata

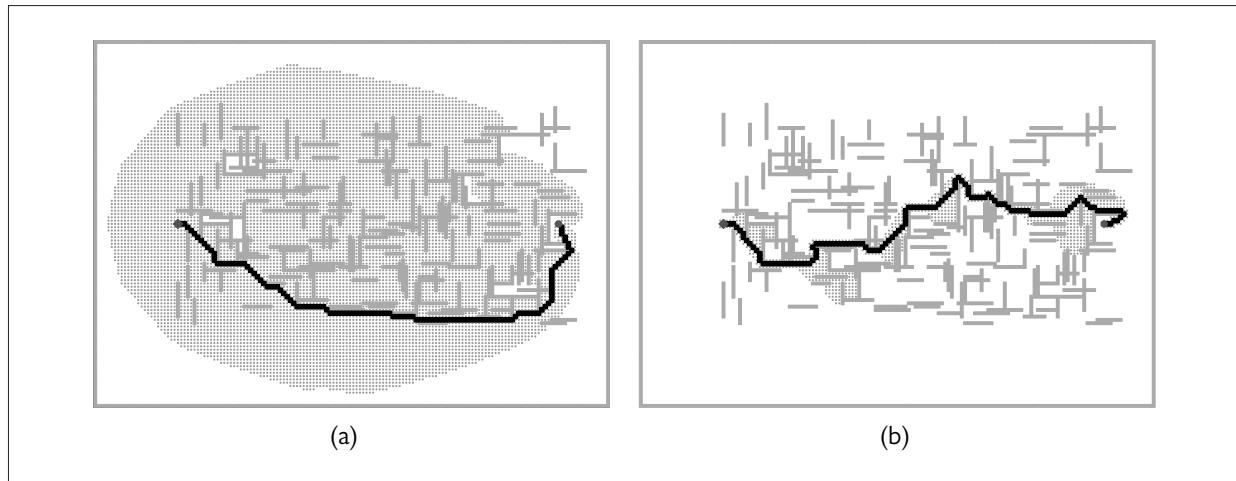


Figura 3.21 Due ricerche sulla stessa griglia: (a) una ricerca A* e (b) una ricerca A* pesata con peso $W = 2$. Le barre in grigio sono ostacoli, la linea nera spessa è il cammino che porta dall'inizio (a sinistra) all'obiettivo (a destra) e i puntini sono gli stati raggiunti da ogni ricerca. In questo particolare problema, la ricerca A* pesata esamina un numero di stati 7 volte inferiore e trova un cammino del 5% più costoso.

La Figura 3.21 mostra un problema di ricerca su una griglia. In (a), una ricerca A* trova la soluzione ottima, ma per ottenerla deve esplorare un'ampia porzione dello spazio degli stati. In (b), una ricerca A* pesata trova una soluzione leggermente più costosa, ma con un tempo di ricerca molto inferiore. Notiamo che la ricerca pesata focalizza il confine degli stati raggiunti verso un obiettivo: ciò significa che vengono esplorati meno stati, ma se il cammino ottimo va al di fuori del confine della ricerca pesata (come avviene in questo caso), il cammino ottimo non viene trovato. In generale, se la soluzione ottima ha costo C^* , una ricerca A* pesata troverà una soluzione dal costo compreso tra C^* e $W \times C^*$; nella pratica, tuttavia, solitamente otteniamo risultati molto più vicini a C^* che a $W \times C^*$. Abbiamo considerato ricerche che valutano gli stati combinando g e h in vari modi; la ricerca A* pesata può essere considerata una generalizzazione delle altre:

Ricerca A [*] :	$g(n) + h(n)$	$(W = 1)$
Ricerca a costo uniforme:	$g(n)$	$(W = 0)$
Ricerca best-first greedy:	$h(n)$	$(W = \infty)$
Ricerca A [*] pesata:	$g(n) + W \times h(n)$	$(1 < W < \infty)$

La ricerca A* pesata si potrebbe anche chiamare “ricerca un po’ golosa”: come la ricerca best-first greedy, focalizza la ricerca verso un obiettivo; d’altra parte, non ignorerà del tutto il costo del cammino: un cammino che realizza scarsi progressi ad alti costi verrà sospeso.

Esistono vari algoritmi di ricerca subottimi, che possono essere caratterizzati in base ai criteri per cui una soluzione è considerata “sufficientemente buona”. Nella **ricerca a sub-ottimalità limitata** cerchiamo una soluzione per cui vi sia la garanzia che si trovi entro un fattore costante W del costo ottimo. La ricerca A^* pesata fornisce tale garanzia. Nella **ricerca a costo limitato** cerchiamo una soluzione il cui costo sia inferiore a una costante C . Nella **ricerca a costo illimitato**, infine, accettiamo una soluzione di qualsiasi costo, purché riusciamo a trovarla rapidamente.

Un esempio di algoritmo di ricerca a costo illimitato è la **ricerca veloce** (*speedy*), una versione della ricerca best-first greedy che utilizza come euristica il numero stimato di azioni richieste per raggiungere un obiettivo, a prescindere dal costo di tali azioni. Per problemi in cui tutte le azioni hanno lo stesso costo, tale ricerca coincide con la ricerca best-first greedy.

- ricerca a subottimalità limitata
- ricerca a costo limitato
- ricerca a costo illimitato
- ricerca veloce

ma quando le azioni hanno costi diversi, tende a condurre la ricerca in modo da trovare rapidamente una soluzione, anche se potrebbe avere un costo elevato.

3.5.5 Ricerca con memoria limitata

La principale problematica della ricerca A* è legata all'impiego della memoria. In questo paragrafo esamineremo alcuni trucchi di implementazione che consentono di risparmiare spazio in memoria e poi alcuni algoritmi interamente nuovi che sfruttano meglio lo spazio disponibile.

La memoria è suddivisa tra la *frontiera* e gli stati *raggiunti*. Nella nostra implementazione della ricerca best-first, uno stato che si trova sulla frontiera è memorizzato in due posizioni: come nodo nella frontiera (così possiamo decidere il prossimo nodo da espandere) e come elemento nella tabella degli stati raggiunti (così sappiamo se lo abbiamo già visitato). Per molti problemi (come quello di esplorare una griglia) questa ridondanza non desta preoccupazioni dato che la dimensione della *frontiera* è molto più piccola della dimensione di *raggiunti*, perciò la duplicazione degli stati nella frontiera richiede una quantità di memoria praticamente trascurabile. Alcune implementazioni, tuttavia, mantengono gli stati in una sola delle due posizioni, risparmiando un po' di spazio in cambio di una maggiore complessità (e forse un rallentamento) dell'algoritmo.

Un'altra possibilità è quella di rimuovere gli stati da *raggiunti* quando si è in grado di dimostrare che non servono più. Per alcuni problemi possiamo usare la proprietà di separazione (Figura 3.6), insieme al divieto di azioni reversibili, per garantire che tutte le azioni muovano verso l'esterno dalla frontiera oppure verso un altro stato di frontiera. In tal caso, è sufficiente controllare se nella frontiera ci sono cammini ridondanti e si può eliminare la tabella *raggiunti*.

Per altri problemi possiamo tenere traccia del numero di volte che uno stato è stato raggiunto e rimuoverlo dalla tabella *raggiunti* quando non vi sono più modi per raggiungerlo. Per esempio, nel caso di un mondo a griglia in cui ogni stato può essere raggiunto soltanto a partire da quelli adiacenti, una volta raggiunto lo stato quattro volte, possiamo rimuoverlo dalla tabella.

Ora consideriamo alcuni nuovi algoritmi progettati per risparmiare spazio in memoria.

La **ricerca beam** (letteralmente “ricerca a fascio”, inteso come fascio di luce) limita la dimensione della frontiera. L’approccio più semplice consiste nel mantenere soltanto i k nodi con i migliori costi f , scartando ogni altro nodo espanso; questo naturalmente rende la ricerca incompleta e subottima, ma possiamo scegliere k per fare buon uso della memoria disponibile, e l’algoritmo viene eseguito velocemente perché espande meno nodi. Per molti problemi questo approccio consente di trovare buone soluzioni vicine all’ottimo. Possiamo visualizzare i meccanismi di ricerca nel seguente modo: mentre la ricerca a costo uniforme o la ricerca A* si espandono in confini concentrici, la ricerca beam esplora soltanto un settore di tali confini, il “fascio” che contiene i k migliori candidati.

Una versione alternativa della ricerca beam non impone un limite stretto sulla dimensione della frontiera, ma mantiene ogni nodo il cui costo f rientra in un fattore δ del migliore costo f . In questo modo, quando ci sono pochi nodi con un buon costo f , ne vengono mantenuti pochi, ma se non ci sono nodi con un buon costo f , ne vengono mantenuti di più fino all’emergere di un nodo con un buon costo f .

La **ricerca A* ad approfondimento iterativo** (IDA*, da *Iterative-Deepening A**) sta alla ricerca A* come la ricerca ad approfondimento iterativo sta alla ricerca in profondità: la ricerca IDA* fornisce i vantaggi della ricerca A* senza la necessità di mantenere in memoria tutti gli stati raggiunti, al costo di visitare alcuni stati più volte. È un algoritmo di ricerca molto importante e usato spesso per problemi che richiederebbero altrimenti troppa memoria.

ricerca beam

ricerca A* ad approfondimento iterativo

```

function RICERCA-BEST-FIRST-RICORSIVA(problema) returns una soluzione o fallimento
  soluzione, valoref  $\leftarrow$  RBFS(problema, NODO(problema.STATOINIZIALE),  $\infty$ )
  return soluzione

function RBFS(problema, nodo, f_limite) returns una soluzione o fallimento e un nuovo limite al costo f
  if problema.È-OBIETTIVO(nodo.STATO) then return nodo
  successori  $\leftarrow$  LISTA(ESPANDI(nodo))
  if successori è vuoto then return fallimento,  $\infty$ 
  for each s in successori do // aggiorna f con il valore ottenuto dalla ricerca precedente
    s.f  $\leftarrow$  max(s.COSTO-CAMMINO + h(s), nodo.f)
  while true do
    migliore  $\leftarrow$  il nodo con valoref minimo in successori
    if migliore.f  $>$  f_limite then return fallimento, migliore.f
    alternativa  $\leftarrow$  il secondo nodo con valoref minimo tra successori
    risultato, migliore.f  $\leftarrow$  RBFS(problema, migliore, min(f_limite, alternativa))
    if risultato  $\neq$  fallimento then return risultato, migliore.f

```

Figura 3.22 Algoritmo per ricerca best-first ricorsiva.

Nell’approfondimento iterativo standard la soglia è la profondità, che aumenta di uno a ogni iterazione. Nella ricerca IDA* la soglia è il costo $f(g + h)$; a ogni iterazione il valore soglia è il più piccolo costo f di qualsiasi nodo che abbia superato la soglia nella precedente iterazione. In altre parole, ogni iterazione esegue una ricerca esaustiva di un confine f , trova un nodo appena al di là di tale confine e utilizza il costo f di tale nodo come confine successivo. Per problemi come il rompicapo a 8 tasselli in cui ogni cammino ha costo f intero, questo algoritmo funziona molto bene, realizzando a ogni iterazione un progresso verso l’obiettivo. Se la soluzione ottimale ha costo C^* , non possono servire più di C^* iterazioni (per esempio, non più di 31 iterazioni sui più difficili rompicapo a 8 tasselli). Per un problema in cui ogni nodo ha un costo f diverso, invece, ogni nuovo confine potrebbe contenere un solo nodo nuovo e il numero di iterazioni potrebbe essere uguale al numero degli stati.

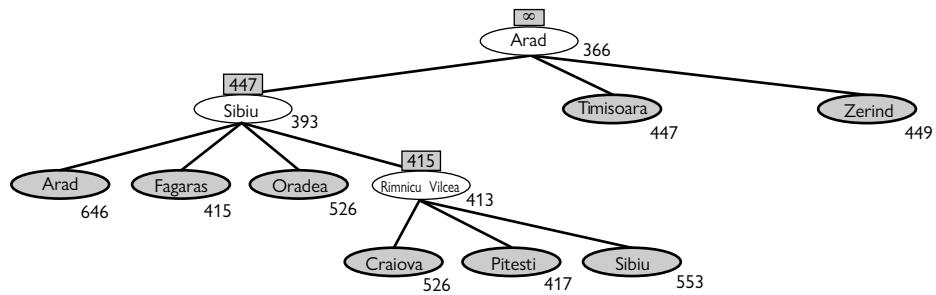
ricerca best-first
ricorsiva

La **ricerca best-first ricorsiva** (RBFS, da *Recursive Best-First Search*), illustrata nella Figura 3.22, tenta di imitare il funzionamento di una ricerca best-first standard usando solamente uno spazio lineare. L’algoritmo assomiglia a una ricerca in profondità ricorsiva, ma invece di continuare a seguire indefinitamente il cammino corrente, utilizza la variabile *f_limite* per tenere traccia del valore *f* del miglior cammino *alternativo* che parte da uno qualsiasi degli antenati del nodo corrente. Se il nodo corrente supera questo limite, la ricorsione torna indietro al cammino alternativo. Durante il ritorno, RBFS sostituisce il valore *f* di ogni nodo lungo il cammino con un **valore di backup**, il miglior valore *f* dei suoi nodi figli. In questo modo RBFS ricorda il valore *f* della foglia migliore nel sottoalbero abbandonato e può quindi decidere in seguito di ri-espanderlo. La Figura 3.23 mostra come RBFS raggiunge Bucarest.

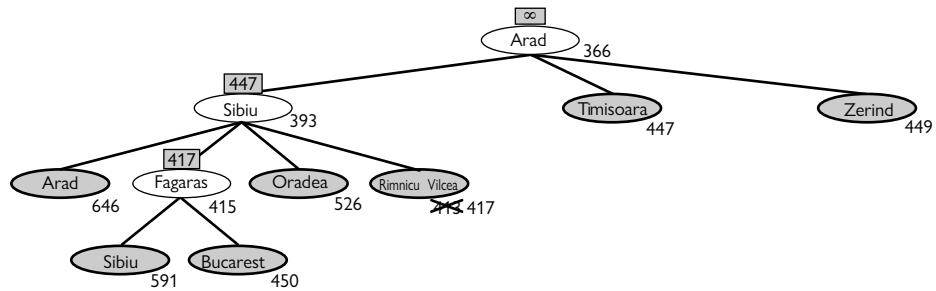
valore di backup

La ricerca RBFS è più efficiente della ricerca IDA*, ma soffre ancora per un’eccessiva rigenerazione di nodi. Nell’esempio della Figura 3.23 la ricerca RBFS segue prima il cammino che passa per Rimnicu Vilcea, poi “ci ripensa” e prova Fagaras, poi cambia ancora idea. Questo si verifica perché, ogni volta che viene esteso il cammino migliore, c’è una buona probabilità che il suo valore *f* aumenti: infatti di solito *h* è meno ottimista quando i nodi sono più vicini a un obiettivo. Quando ciò accade, e in particolar modo quando gli spazi di ricerca sono grandi, il secondo cammino migliore può diventare quello migliore, e così la ri-

(a) Dopo l'espansione di Arad, Sibiu e Rimnicu Vilcea



(b) Dopo essere tornati indietro a Sibiu e aver espanso Fagaras



(c) Dopo essere ritornati ancora a Rimnicu Vilcea e aver espanso Pitesti

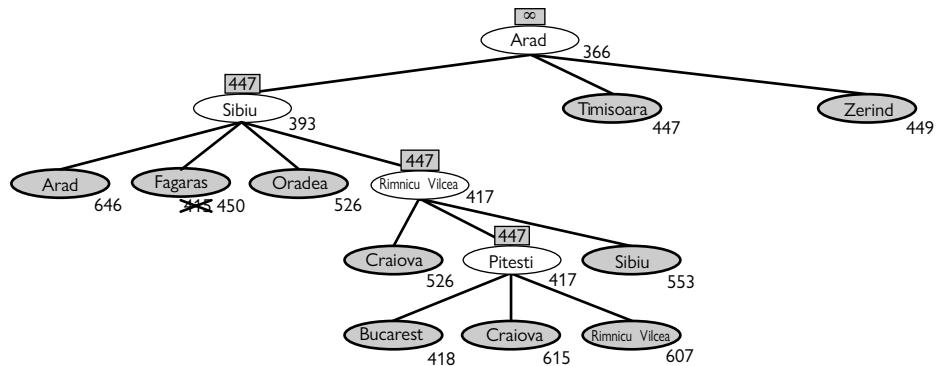


Figura 3.23 Fasi di una ricerca RBFS per trovare l'itinerario più breve verso Bucarest. Il limite ai valori f di ogni chiamata ricorsiva è indicato in un box sopra al nodo corrente, e ogni nodo è etichettato con il suo costo f . (a) Si segue il cammino che passa da Rimnicu Vilcea finché la foglia migliore corrente (Pitesti) non ha un valore peggiore del miglior cammino alternativo (Fagaras). (b) La ricorsione “torna su” e il valore della foglia migliore del sottoalbero dimenticato (417) è salvato in Rimnicu Vilcea; quindi viene espanso Fagaras, rivelando che il valore della foglia migliore è 450. (c) La ricorsione torna ancora indietro e il valore della foglia migliore del sottoalbero dimenticato (450) è salvato in Fagaras; quindi si espande Rimnicu Vilcea. Stavolta, dato che il miglior cammino alternativo (quello che passa per Timisoara) costa almeno 447, l'espansione continua fino a raggiungere Bucarest.

cerca deve tornare indietro per seguirlo. Ogni “ripensamento” corrisponde a un’iterazione di IDA* e può richiedere molte ri-expansioni di nodi dimenticati per ricreare il cammino migliore ed estenderlo di un nodo.

RBFS è un algoritmo ottimo se la funzione euristica $h(n)$ è ammissibile. La sua complessità spaziale è lineare nella profondità della più profonda soluzione ottimale, ma quella temporale è abbastanza difficile da definire, perché dipende sia dall'accuratezza della funzione euristica sia dalla frequenza dei cambiamenti del cammino ottimale durante l'espansione dei nodi. L'algoritmo RBFS espande i nodi in ordine crescente di costo f anche se f non è monotona.

Il problema degli algoritmi IDA* e RBFS è che usano *trop poco* memoria. Tra un'iterazione e l'altra, IDA* ricorda solo un numero: il limite corrente all' f -costo. RBFS mantiene in memoria più informazione, ma usa solo uno spazio lineare: anche se fosse disponibile molta memoria di più, non la userebbe. Poiché entrambi gli algoritmi dimenticano la maggior parte di ciò che hanno fatto, potrebbero riesplorare nuovamente gli stessi stati molte volte.

Sembra quindi importante determinare quanta memoria è disponibile e consentire a un algoritmo di utilizzarla tutta. Due algoritmi che lo fanno sono **MA*** (*memory-bounded A**) e **SMA*** (*simplified MA**). Tra i due descriveremo SMA*, che come dice il nome è una versione semplificata del primo. SMA* procede proprio come A*, espandendo la foglia migliore finché la memoria è piena. A questo punto non può aggiungere un nuovo nodo all'albero di ricerca senza cancellarne uno vecchio. SMA* scarta sempre il nodo foglia *peggiore*, quello con costo f più alto. Come RBFS, memorizza nel nodo padre il valore del nodo dimenticato. In questo modo la radice di un sottoalbero dimenticato conosce la qualità del cammino migliore in quel sottoalbero. Con quest'informazione SMA* ri-genera il sottoalbero dimenticato solo quando *tutti gli altri cammini* promettono di comportarsi peggio di quello. Potremo anche dire che, se tutti i discendenti di un nodo n sono dimenticati, allora non sappiamo da che parte andare partendo da n , ma abbiamo ancora un'idea chiara di quanto sia conveniente passare per n .

L'algoritmo completo è riportato nel codice disponibile online nel sito originale degli autori. Vale comunque la pena di menzionare una sottigliezza. Abbiamo detto che SMA* espande la foglia migliore e cancella quella peggiore. Ma cosa succede se *tutte* le foglie hanno lo stesso costo f ? Per evitare di scegliere lo stesso nodo sia per la cancellazione che per l'espansione, SMA* espande la foglia migliore *più recente* e cancella la foglia peggiore *più vecchia*. Questi due nodi coincidono quando c'è una sola foglia, ma in tal caso l'albero di ricerca deve consistere in un singolo cammino, che riempie tutta la memoria, dalla radice alla foglia in questione. Se la foglia non è un nodo obiettivo, allora *anche se si trovasse sul cammino di una soluzione ottima* tale soluzione non sarebbe raggiungibile con la memoria disponibile. Di conseguenza quel nodo può essere scartato esattamente come se non avesse successori.

L'algoritmo SMA* è completo se c'è una soluzione raggiungibile, ovvero se d , la profondità del nodo obiettivo più vicino alla radice, è inferiore alla dimensione della memoria espressa in nodi. La strategia è ottima se c'è una soluzione ottima raggiungibile; altrimenti viene restituita la soluzione migliore tra quelle raggiungibili. In pratica, SMA* è una scelta piuttosto affidabile per trovare soluzioni ottime, in particolare quando lo spazio degli stati è un grafo, i costi di azione non sono uniformi e la generazione dei nodi è costosa rispetto all'elaborazione richiesta per il mantenimento della frontiera e dell'insieme degli stati raggiunti.

Nel caso di problemi molto difficili, comunque, una ricerca SMA* sarà spesso obbligata a passare continuamente dall'uno all'altro tra molti cammini candidati, di cui sarà possibile tenere in memoria solo un piccolo sottoinsieme (questo problema ricorda quello del **trashing** nei sistemi operativi con paginazione su disco). A questo punto il tempo aggiuntivo richiesto per la generazione ripetuta degli stessi nodi potrebbe far sì che problemi risolvibili in pratica da A*, a patto di avere memoria illimitata, diventino intrattabili per SMA*. Questo significa che *le limitazioni di memoria possono rendere un problema intrattabile sotto l'aspetto temporale*. Sebbene non ci sia una teoria che formalizzi il problema del compromesso tra tempo e memoria, non sembra che esista una soluzione efficace: l'unica possibilità è rinunciare al requisito dell'ottimalità.

MA*
SMA*

trashing



3.5.6 Ricerca euristica bidirezionale

Nel caso della ricerca best-first unidirezionale abbiamo visto che usando $f(n) = g(n) + h(n)$ come funzione della valutazione otteniamo una ricerca A* con la garanzia di trovare soluzioni ottime rispetto al costo (assumendo una h ammissibile) e ottimamente efficiente nel numero di nodi espanso.

Potremmo provare a usare $f(n) = g(n) + h(n)$ anche con la ricerca best-first bidirezionale, ma purtroppo in questo caso non vi è garanzia di arrivare a una soluzione ottima, né che la ricerca sia ottimamente efficiente, anche con un'euristica ammissibile. Nel caso della ricerca bidirezionale risulta che non sono i singoli nodi, ma le *coppie* di nodi (uno da ciascuna frontiera) che possiamo dimostrare che vengano espansi con certezza, perciò qualsiasi dimostrazione di efficienza dovrà considerare coppie di nodi (Eckerle *et al.*, 2017).

Introduciamo ora una nuova notazione. Utilizziamo $f_F(n) = g_F(n) + h_F(n)$ per nodi nella direzione in avanti (con lo stato iniziale come radice) e $f_B(n) = g_B(n) + h_B(n)$ per nodi nella direzione all'indietro (con uno stato obiettivo come radice). Entrambe le ricerche in avanti e all'indietro mirano a risolvere lo stesso problema, ma hanno funzioni di valutazioni diverse perché, per esempio, le euristiche sono diverse in base al fatto che si punti a trovare l'obiettivo o lo stato iniziale. Assumeremo euristiche ammissibili.

Consideriamo un cammino in avanti dallo stato iniziale a un nodo m e un cammino all'indietro dall'obiettivo a un nodo n . Possiamo definire un limite inferiore sul costo di una soluzione che segue il cammino dallo stato iniziale a m , poi arriva a n , poi segue il cammino che porta all'obiettivo come:

$$lb(m, n) = \max(g_F(m) + g_B(n), f_F(m), f_B(n)).$$

In altre parole, il costo di tale cammino deve essere almeno pari alla somma dei costi di cammino delle due parti (perché la rimanente connessione tra di essi deve avere costo non negativo) e tale costo deve anche essere almeno pari al costo f stimato di ciascuna delle due parti (perché le stime fornite da euristiche sono ottimistiche). Dato ciò, il teorema afferma che, per ogni coppia di nodi m, n con $lb(m, n)$ minore del costo ottimale C^* , dobbiamo espandere m o n , perché il cammino che passa per entrambi è una potenziale soluzione ottimale. La difficoltà sta nel fatto che non conosciamo con certezza quale nodo è meglio espandere, perciò nessun algoritmo di ricerca bidirezionale può dare garanzia di essere ottimamente efficiente – qualsiasi algoritmo potrebbe espandere fino a due volte il numero minimo di nodi se sceglie sempre l'elemento sbagliato di una coppia da espandere per primo. Alcuni algoritmi di ricerca euristica bidirezionale gestiscono esplicitamente una coda di coppie (m, n) , ma noi ci limiteremo alla ricerca best-first bidirezionale (Figura 3.14), che ha due code con priorità come frontiere, e le forniremo una funzione di valutazione che imita il criterio lb :

$$f_2(n) = \max(2g(n), g(n) + h(n)).$$

Il prossimo nodo da espandere sarà quello che minimizza questo valore f_2 , e può provenire da entrambe le frontiere. Questa funzione f_2 garantisce che non espanderemo mai un nodo (da qualsiasi frontiera) con $g(n) > \frac{C}{2}$. Diciamo che le due metà della ricerca “si incontrano a metà strada” nel senso che, quando le due frontiere entrano in contatto, nessun nodo al loro interno ha un costo di cammino maggiore del valore limite $\frac{C}{2}$. La Figura 3.24 mostra come procede un esempio di ricerca bidirezionale.

Abbiamo descritto un approccio in cui l'euristica h_F stima la distanza dall'obiettivo (o, quando il problema ha più stati obiettivo, la distanza dall'obiettivo più vicino) e h_B stima la distanza dall'inizio; in questo caso parliamo di **ricerca front-to-end**. Un'alternativa, la **ricerca front-to-front**, cerca di stimare la distanza dall'altra frontiera. Naturalmente con una frontiera di milioni di nodi sarebbe inefficiente applicare la funzione euristica a ognuno di essi e calcolare il minimo, ma può funzionare la strategia di prendere un campione di alcuni nodi della

ricerca front-to-end
ricerca front-to-front

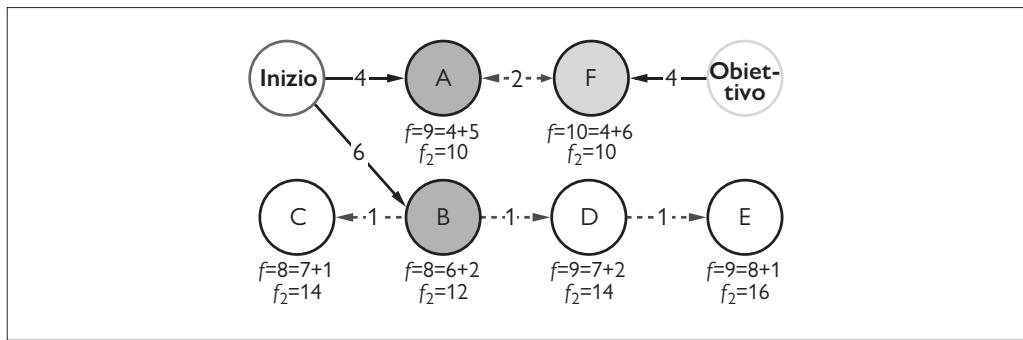


Figura 3.24 La ricerca bidirezionale mantiene due frontiere: sulla sinistra i nodi A e B sono successori di Inizio; sulla destra il nodo F è un successore inverso di Obiettivo. Ciascun nodo è etichettato con $f = g + h$ valori e il valore $f_2 = \max(2g, g + h)$ (i valori g sono la somma dei costi di azione mostrati su ciascuna freccia; i valori h sono arbitrari e non possono essere ricavati da alcun elemento della figura). La soluzione ottimale, Inizio-A-F-Obiettivo, ha costo $C^* = 4 + 2 + 4 = 10$, perciò un algoritmo bidirezionale con un “incontro a metà strada” non dovrebbe espandere alcun nodo con $g > C^* = 5$; infatti, il prossimo nodo da espandere dovrebbe essere A o F (entrambi con $g = 4$), che ci porterebbe a una soluzione ottimale. Se espandessimo prima il nodo con costo f minimo, i prossimi sarebbero B e C, e D ed E sarebbero collegati con A, ma dato che questi nodi hanno tutti $g > C^*/2$, non vengono mai espansi quando la funzione di valutazione è f_2 .

frontiera. In alcuni specifici domini di problemi è possibile *riassumere* la frontiera – per esempio, in un problema di ricerca su griglia possiamo calcolare in modo incrementale un riquadro che racchiude la frontiera e usare come euristica la distanza da tale riquadro.

La ricerca bidirezionale a volte è più efficiente di quella unidirezionale, a volte no. In genere, se abbiamo un’euristica molto buona, una ricerca A^* produce confini focalizzati sull’obiettivo, e aggiungere una ricerca bidirezionale non è di grande aiuto. Con un’euristica media, invece, una ricerca bidirezionale con “incontro a metà strada” tende a espandere meno nodi e viene preferita. Nel caso peggiore di un’euristica scadente, la ricerca non è più focalizzata sull’obiettivo e la ricerca bidirezionale ha la stessa complessità asintotica di A^* . Una ricerca bidirezionale con funzione di valutazione f_2 e un’euristica ammissibile h è completa e ottima.

3.6 Funzioni euristiche

In questo paragrafo esaminiamo come l’accuratezza di un’euristica influisca sulla prestazione della ricerca e vediamo anche come sia possibile inventare euristiche. Come riferimento principale torneremo all’esempio del rompicapo a 8 tasselli. Come si è detto nel Paragrafo 3.2, lo scopo del rompicapo è di far scivolare i tasselli orizzontalmente o verticalmente nello spazio vuoto, finché la configurazione del gioco corrisponde a quella obiettivo (Figura 3.25).

Ci sono $9!/2 = 181.400$ stati raggiungibili in un rompicapo a 8 tasselli, perciò una ricerca potrebbe facilmente mantenerli in memoria tutti. Nel caso di un rompicapo a 15 tasselli, tuttavia, ci sono $16!/2$ stati – più di 10.000 miliardi – perciò una ricerca in tale spazio richiederà l’aiuto di una buona funzione euristica ammissibile. C’è un lunga storia di tali euristiche per il rompicapo a 15 tasselli; descriviamo qui due delle candidate più comuni:

- h_1 = il numero di tasselli fuori posto (spazi vuoti non inclusi). Nella Figura 3.25 gli otto tasselli sono tutti fuori posto, per cui lo stato iniziale ha $h_1 = 8$. h_1 è un’euristica ammissibile, perché ogni tassello fuori posto richiederà almeno uno spostamento per farlo arrivare al posto giusto;

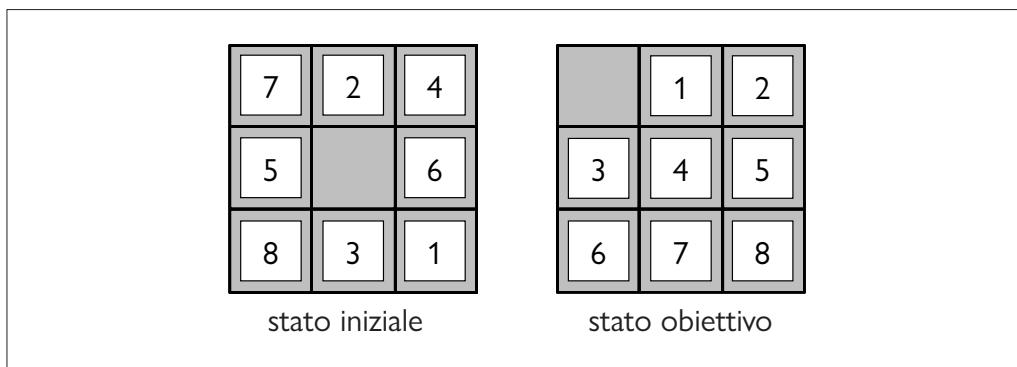


Figura 3.25
Un'istanza tipica del rompicapo a 8 tasselli.
La soluzione più breve è lunga 26 azioni.

- h_2 = la somma delle distanze di tutti i tasselli dalla loro posizione corrente a quella nella configurazione obiettivo. Dato che i tasselli non si possono muovere in diagonale, la distanza è la somma delle distanze in orizzontale e verticale, talvolta chiamata **distanza tra isolati** o **distanza Manhattan**. Anche h_2 è ammissibile, perché una mossa può al massimo spostare un tassello un passo più vicino al suo obiettivo. Considerando i tasselli nell'ordine da 1 a 8, nello stato iniziale della Figura 3.25 abbiamo una distanza Manhattan pari a:

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

Come speravamo, nessuna di queste due euristiche sovrastima il vero costo della soluzione, che è 26.

distanza Manhattan

3.6.1 Effetto dell'accuratezza dell'euristica sulle prestazioni

Un modo di caratterizzare la qualità di un'euristica è il **fattore di ramificazione effettivo**, indicato con b^* . Se il numero totale di nodi generati da A^* per un particolare problema è N , e la profondità è d , allora b^* è il fattore di ramificazione che un albero uniforme di profondità d dovrebbe avere per contenere $N + 1$ nodi. Quindi,

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

Per esempio, se A^* trova una soluzione a profondità 5 usando 52 nodi, il fattore di ramificazione effettivo è 1,92. Questo fattore può cambiare da un'istanza del problema all'altra, ma solitamente per un dominio del problema specifico (come quello dei rompicapi a 8 tasselli) ha un valore abbastanza costante su tutte le istanze di problemi non banali. Di conseguenza, misurazioni sperimentali di b^* su un numero abbastanza piccolo di problemi può già fornire una buona idea dell'utilità generale dell'euristica. Un'euristica ben progettata dovrebbe avere un valore di b^* vicino a 1, permettendo così la soluzione di problemi abbastanza grandi a un costo computazionale ragionevole.

Korf e Reid (1998) sostengono che un modo migliore per caratterizzare l'effetto della potatura A^* con una data euristica h è dire che riduce la **profondità effettiva** di una costante k_h rispetto alla profondità reale. Questo significa che il costo totale della ricerca è $O(b^{d-k_h})$, mentre per una ricerca non informata è $O(b^d)$. Gli esperimenti svolti sul cubo di Rubik e sui problemi di rompicapo a n tasselli dimostrano che tale formula fornisce previsioni accurate del costo di ricerca totale per istanze di problemi campionate su un ampio intervallo di lunghezze della soluzione, almeno per lunghezze maggior di k_h .

fattore di ramificazione effettivo

Per ottenere la Figura 3.26 abbiamo generato a caso numerosi problemi di rompicapo a 8 tasselli e li abbiamo risolti con una ricerca in ampiezza non informata e con una ricerca A^* usando sia h_1 che h_2 ; abbiamo poi riportato il numero medio di nodi generati e il fattore di ramificazione effettivo corrispondente a ogni strategia di ricerca e per diverse lunghezze del-

profondità effettiva

d	Costo di ricerca (nodi generati)			Fattore di ramificazione effettivo		
	RIA	$A^*(h_1)$	$A^*(h_2)$	RIA	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2,01	1,42	1,34
8	368	48	31	1,91	1,40	1,30
10	1033	116	48	1,85	1,43	1,27
12	2672	279	84	1,80	1,45	1,28
14	6783	678	174	1,77	1,47	1,31
16	17270	1683	364	1,74	1,48	1,32
18	41558	4102	751	1,72	1,49	1,34
20	91493	9905	1318	1,69	1,50	1,34
22	175921	22955	2548	1,66	1,50	1,34
24	290082	53039	5733	1,62	1,50	1,36
26	395355	110372	10080	1,58	1,50	1,35
28	463234	202565	22055	1,53	1,49	1,36

Figura 3.26 Confronto dei costi di ricerca e dei fattori di ramificazione effettivi per problemi di rompicapo a 8 tasselli con utilizzo di ricerca in ampiezza, A^* con h_1 (tasselli fuori posto) e A^* con h_2 (distanza Manhattan). I dati sono mediati su 100 istanze per ogni lunghezza della soluzione d da 6 a 28.

la soluzione. I risultati indicano che h_2 è migliore di h_1 ed entrambe sono migliori rispetto a non avere alcuna euristica.

dominazione

Ci si potrebbe chiedere se h_2 sia *sempre* migliore di h_1 ; la risposta è: “In sostanza, sì”. Dalle definizioni delle due euristiche è facile vedere che, per ogni nodo n , $h_2(n) \geq h_1(n)$. Diciamo quindi che h_2 **domina** h_1 . La dominazione si traduce direttamente in efficienza: la ricerca A^* che usa h_2 non espanderà mai più nodi di quella che usa h_1 (eccetto casi sfortunati di rottura di pareggi). La dimostrazione è semplice: ricordate l’osservazione fatta nel Paragrafo 3.5.3 per cui ogni nodo con $f(n) < C^*$ sarà sicuramente espanso. Questo è equivalente a dire che sarà sicuramente espanso ogni nodo con $h(n) < C^* - g(n)$ quando h è consistente. Ma, dato che h_2 è grande almeno quanto h_1 per tutti i nodi, ogni nodo espanso dalla ricerca A^* che usa h_2 lo sarà anche con h_1 , e anche h_1 potrà causare l’espansione di altri nodi. Se ne deduce che è generalmente meglio usare una funzione euristica con valori più alti, a patto che sia consistente e che il tempo per calcolarla non sia troppo lungo.

problema rilassato

3.6.2 Generare euristiche da problemi rilassati

Abbiamo visto che sia h_1 (tasselli fuori posto) che h_2 (distanza Manhattan) sono euristiche abbastanza buone per il rompicapo a 8 tasselli, e che tra le due è preferibile h_2 . Ma com’è possibile inventare un’euristica come h_2 ? Un computer può farlo automaticamente?

h_1 e h_2 sono stime della lunghezza del cammino rimanente che conduce alla soluzione, ma sono anche lunghezze di cammino perfettamente accurate per versioni *semplificate* del rompicapo. Se le regole del problema fossero cambiate in modo che un tassello potesse muoversi ovunque, invece che solo nello spazio vuoto adiacente, h_1 rappresenterebbe la lunghezza esatta della soluzione più breve. In modo analogo, se un tassello potesse muoversi di uno spazio in ogni direzione, anche se la posizione fosse già occupata da un altro tassello, h_2 indicherebbe la lunghezza esatta della soluzione ottimale. Un problema con meno restrizioni sulle azioni possibili è detto **problema rilassato**. Il grafo dello spazio degli stati per il problema rilassato è un *supergrafo* di quello dello spazio degli stati originale, perché la rimozione dei vincoli crea nuovi archi nel grafo.



Poiché il problema rilassato aggiunge archi al grafo dello spazio degli stati, qualsiasi soluzione ottima del problema originale è, per definizione, soluzione anche del problema rilassato, che però può avere soluzioni *migliori* se gli archi aggiunti forniscono delle scorciatoie. Quindi, *il costo di una soluzione ottima di un problema rilassato è un'euristica ammissibile per il problema originale*. Inoltre, poiché l'euristica derivata è un costo esatto per il problema rilassato, deve rispettare la disugualanza triangolare ed è quindi consistente (cfr. Paragrafo 3.5.2).

Se la definizione di un problema è scritta in un linguaggio formale, è possibile costruire automaticamente i suoi rilassamenti.¹⁴ Per esempio, se le azioni del rompicapo sono descritte come segue:

Un tassello si può muovere dalla posizione X a quella Y se
X è adiacente orizzontalmente o verticalmente a Y **and** Y è vuota

possiamo generare tre problemi rilassati rimuovendo una o entrambe le condizioni:

- (a) un tassello può muoversi da X a Y se X e Y sono adiacenti orizzontalmente o verticalmente;
- (b) un tassello può muoversi da X a Y se Y è vuota;
- (c) un tassello può muoversi da X a Y.

Da (a) possiamo derivare h_2 (distanza Manhattan). L'idea è che h_2 sarebbe la distanza esatta dalla soluzione se potessimo semplicemente muovere i tasselli uno dopo l'altro fino alla loro destinazione. L'euristica derivata da (b) è discussa nell'Esercizio 3.GASC. Da (c) possiamo derivare h_1 (tasselli fuori posto), perché tale euristica rappresenterebbe la distanza esatta se i tasselli potessero essere spostati alla loro destinazione in una sola azione. Un aspetto cruciale è che i problemi rilassati generati con questa tecnica possono essere risolti praticamente *senza ricerca*, perché il rilassamento delle regole permette di scomporre il problema in otto sottoproblemi indipendenti. Se il problema rilassato è ancora difficile da risolvere i valori della corrispondente euristica saranno costosi da ottenere.

Esiste un programma chiamato ABSOLVER che può generare automaticamente euristiche partendo dalla definizione dei problemi, usando il metodo dei problemi rilassati e varie altre tecniche (Prieditis, 1993). ABSOLVER ha generato una nuova euristica per il rompicapo a 8 tasselli migliore di tutte quelle preesistenti e ha trovato la prima euristica utile per il famoso cubo di Rubik.

Se per un problema abbiamo una collezione di euristiche ammissibili $h_1 \dots h_m$ e nessuna domina le altre, quale dovremmo scegliere? Possiamo prendere il meglio di tutte definendo:

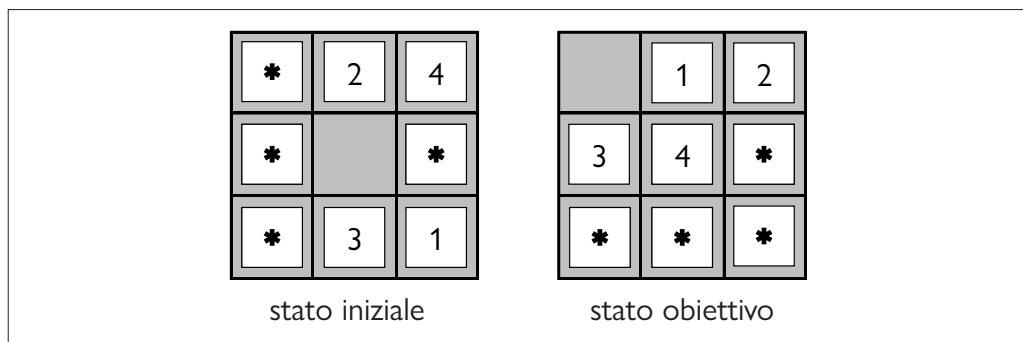
$$h(n) = \max\{h_1(n), \dots, h_m(n)\}.$$

Questa euristica composita sceglie la funzione più accurata per ogni nodo. Dato che tutte le euristiche componenti sono ammissibili, h è ammissibile (e se le h_i sono tutte consistenti, h è consistente). L'unico svantaggio è che $h(n)$ richiede più tempo di calcolo. Se questo costituisce un problema, un'alternativa è quella di selezionare a caso una delle euristiche a ogni valutazione, o usare un algoritmo di apprendimento automatico per predire quale sarà l'euristica migliore. In questo modo si può ottenere un'euristica inconsistente (perfino se ogni h_i è consistente), ma nella pratica si ottiene solitamente una risoluzione del problema più rapida.

¹⁴ Nei Capitoli 8 e 11 descrivremo linguaggi formali adatti a questo compito; visto che le descrizioni formali possono essere manipolate, la costruzione di problemi rilassati può essere automatizzata. Per adesso utilizzeremo il linguaggio naturale.

Figura 3.27

Un sottoproblema dell’istanza del rompicapo a 8 tasselli della Figura 3.25. Lo scopo è mettere i tasselli 1, 2, 3, 4 e quello vuoto nella posizione corretta, senza preoccuparsi degli altri.



sottoproblema

3.6.3 Generare euristiche da sottoproblemi: database di pattern

È anche possibile derivare euristiche ammissibili dal costo della soluzione di un **sottoproblema** del problema dato. Per esempio, la Figura 3.27 mostra un sottoproblema dell’istanza del rompicapo della Figura 3.25, dove il sottoproblema considera solo i tasselli 1, 2, 3 e 4 e lo spazio vuoto nelle loro posizioni corrette. Palesemente, il costo della soluzione ottimale di questo sottoproblema è un limite inferiore del costo del problema completo. Quest’euristica, in alcuni casi, si è rivelata molto più accurata della distanza Manhattan.

database di pattern

L’idea alla base dei **database di pattern** è di memorizzare i costi esatti delle soluzioni di ogni possibile istanza di sottoproblema: nel nostro esempio, ogni possibile configurazione dei primi quattro tasselli e dello spazio vuoto. Nel database ci saranno $9 \times 8 \times 7 \times 6 \times 5 = 15.120$ pattern. Le identità degli altri quattro tasselli sono irrilevanti per risolvere il sottoproblema, ma le loro mosse continuano a contare ai fini del calcolo del costo. Ora possiamo calcolare un’euristica ammissibile h_{DB} per ogni stato incontrato durante una ricerca, semplicemente estraendo dal database la corrispondente configurazione del sottoproblema. Il database stesso è costruito eseguendo una ricerca all’indietro dallo stato obiettivo e memorizzando il costo di ogni nuova configurazione incontrata;¹⁵ la spesa sostenuta per questa ricerca è ammortizzata dal fatto che il database può essere usato per molte istanze del problema.

La scelta di usare i primi quattro tasselli e lo spazio vuoto è arbitraria; avremmo potuto costruire database per i tasselli 5-6-7-8, oppure 2-4-6-8, e così via. Ognuno di essi fornisce un’euristica ammissibile, e queste possono essere tutte combinate, come abbiamo visto poco fa, prendendo sempre il loro valore massimo. Un’euristica combinata di questo tipo è molto più accurata della distanza Manhattan; il numero di nodi generati per risolvere un rompicapo a 15 tasselli può essere ridotto anche di un fattore 1000. Tuttavia, con ogni database in più i vantaggi diminuiscono, e aumentano i costi in termini di memoria e di calcolo.

Ci si potrebbe chiedere se sia possibile *sommare* l’euristica ottenuta dal database 1-2-3-4 e quella del database 5-6-7-8, dato che i due sottoproblemi non sembrano sovrapporsi. L’euristica risultante sarebbe ancora ammissibile? La risposta è no, perché le soluzioni del sottoproblema 1-2-3-4 e quelle del sottoproblema 5-6-7-8 in un dato stato condivideranno certamente alcune mosse: in altre parole, è molto improbabile che i tasselli 1-2-3-4 possano essere spostati al loro posto senza toccare i tasselli 5-6-7-8 e viceversa. E se non contassimo quelle mosse? E se non utilizzassimo asterischi quali astrazioni degli altri tasselli, ma li facessimo scomparire? Possiamo considerare non già il costo totale della soluzione del sotto-

¹⁵ Procedendo a ritroso a partire dall’obiettivo, il costo esatto della soluzione di ogni istanza incontrata è immediatamente disponibile. Questo è un esempio di **programmazione dinamica**, che discuteremo meglio nel Capitolo 17.

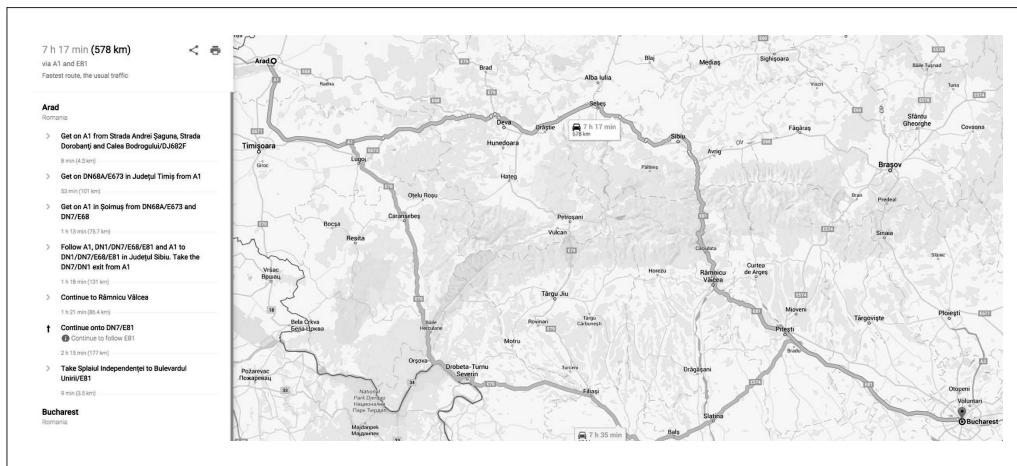


Figura 3.28
Un servizio web che fornisce indicazioni stradali, calcolate da un algoritmo di ricerca.

problema 1-2-3-4, ma solo il numero di mosse che coinvolgono i primi quattro tasselli. Allora la somma dei due costi è ancora un limite inferiore del costo della soluzione dell'intero problema. Questa è l'idea alla base dei **database di pattern disgiunti**. Con questi ultimi, è possibile risolvere istanze casuali del rompicapo a 15 tasselli in pochi millisecondi – il numero di nodi generati è ridotto di un fattore 10.000 rispetto all'uso della distanza Manhattan. Nel caso del rompicapo a 24 tasselli, il processo per trovare una soluzione può essere velocizzato di un fattore di circa un milione. I database di pattern disgiunti funzionano bene per i rompicapo a tasselli, perché dato che si sposta comunque un solo tassello per volta, il problema può essere suddiviso in modo tale che ogni mossa influisce su un solo sottoproblema.

database di pattern disgiunti

3.6.4 Generare euristiche con punti di riferimento (landmark)

Esistono servizi online che ospitano mappe con decine di milioni di vertici e sono in grado di trovare indicazioni stradali per la guida in automobile ottime rispetto al costo in tempi dell'ordine dei millisecondi (Figura 3.28). Come possono farlo, quando i migliori algoritmi di ricerca che abbiamo esaminato finora sono circa un milione di volte più lenti? Esistono molte tecniche, ma la più importante è il **precalcolo** di alcuni costi di cammino ottimi. Anche se il precalcolo può richiedere tempo, va svolto soltanto una volta e in seguito può essere ammortizzato su miliardi di richieste effettuate dagli utenti.

precalcolo

Potremmo generare un'euristica perfetta precalcolando e memorizzando il costo del cammino ottimo tra ogni coppia di vertici. Ciò richiederebbe uno spazio $O(|V|^2)$ e un tempo $O(|E|^3)$, valori sensati per grafi con 10 mila vertici, ma non 10 milioni.

punto di riferimento

Un approccio migliore consiste nello scegliere alcuni (magari 10 o 20) **punti di riferimento** (*landmark*)¹⁶ tra i vertici, e poi per ciascun punto di riferimento L e per ogni altro vertice v nel grafo calcolare e memorizzare $C^*(v, L)$, il costo esatto del cammino ottimo da v a L (ci serve anche $C^*(L, v)$, che su un grafo non orientato coincide con $C^*(v, L)$, mentre su un grafo orientato, per esempio con strade a senso unico, va calcolato separatamente). Disponendo delle tabelle di C^* memorizzate, possiamo facilmente creare un'euristica efficiente (anche se inammissibile): il costo minimo, su tutti i punti di riferimento, per andare dal nodo corrente al punto di riferimento e poi all'obiettivo:

$$h_L(n) = \min_{L \in \text{Riferimenti}} C^*(n, L) + C^*(L, obiettivo)$$

¹⁶ I punti di riferimento sono detti anche “pivot” o “ancore”.

Se il cammino ottimo passa per un punto di riferimento, questa euristica sarà esatta; se no, è inammissibile – sovrasta il costo per raggiungere l'obiettivo. In una ricerca A*, se si hanno euristiche esatte, una volta raggiunto un nodo che si trova su un cammino ottimo, ogni nodo espanso da lì in poi sarà anch'esso su un cammino ottimo. Le linee di confine seguono questo cammino ottimo. La ricerca seguirà quindi il cammino ottimo, aggiungendo a ogni iterazione un'azione con costo c per arrivare a uno stato risultato il cui valore h diminuirà di c , a indicare che il valore totale $f = g + h$ rimarrà costante e uguale a C^* lungo l'intero cammino.

scorciatoia

Alcuni algoritmi per la ricerca di itinerari riescono a risparmiare ancora più tempo aggiungendo delle **scorciatoie**, archi artificiali nel grafo che definiscono un cammino ottimale di più azioni. Per esempio, se vi fossero scorciatoie predefinite tra tutte le 100 più grandi città degli Stati Uniti, e volessimo andare dal campus di Berkeley in California alla New York University a New York, potremmo prendere la scorciatoia tra Sacramento e Manhattan e coprire il 90% del cammino con un'unica azione.

$h_L(n)$ è efficiente ma non ammissibile, ma con un po' più di lavoro possiamo arrivare a un'euristica efficiente e anche ammissibile:

$$h_{ED}(n) = \max_{L \in \text{Riferimenti}} |C^*(n, L) - C^*(\text{obiettivo}, L)|$$

euristica differenziale

Questa è detta **euristica differenziale** (a causa della sottrazione). Pensate a che cosa accade con un punto di riferimento che si trova oltre l'obiettivo. Se l'obiettivo si trova sul cammino ottimo da n al punto di riferimento, l'euristica dice: “Considera l'intero cammino da n a L , poi sottrai l'ultimo pezzo che va da *obiettivo* a L , per ottenere il costo esatto del cammino da n a *obiettivo*”. Se l'obiettivo è un po' fuori del cammino ottimo per raggiungere il punto di riferimento, l'euristica sarà inesatta, ma comunque ammissibile. I punti di riferimento che non sono oltre l'obiettivo non sono utili; con un punto di riferimento posto esattamente a metà strada tra n e *obiettivo* avremo $h_{ED} = 0$, che non ci serve.

I punti di riferimento si possono scegliere in diversi modi. Selezionarli a caso è un modo veloce, ma si ottengono risultati migliori se si presta attenzione a fare in modo che non siano troppo vicini tra di loro. C'è anche un approccio *greedy*: si sceglie un primo punto di riferimento a caso, poi si trova il punto più lontano da esso e lo si aggiunge all'insieme dei punti di riferimento, e si continua così, aggiungendo a ogni iterazione il punto che massimizza la distanza dal punto di riferimento più vicino. Se si sono registrate in un file di log le richieste di ricerca effettuate in passato dagli utenti, è possibile scegliere i punti di riferimento più richiesti. Per l'euristica differenziale è meglio se i punti di riferimento sono diffusi attorno al perimetro del grafo. Quindi, una buona tecnica è quella di trovare il centroide del grafo, disporre attorno a esso k triangoli, come fossero fette di torta, e in ognuno selezionare il vertice più lontano dal centro.

L'uso di punti di riferimento funziona bene in particolare nei problemi di ricerca di itinerari stradali, a causa del modo in cui vengono costruite le strade: poiché c'è sempre tanto traffico tra i punti di riferimento, le strade di collegamento tra questi sono le più ampie e veloci, e la ricerca con punti di riferimento facilita l'individuazione di queste strade.

3.6.5 Imparare a cercare meglio

spazio degli stati di metalivello

Abbiamo presentato diverse strategie di ricerca – ricerca in ampiezza, A* e così via – che sono state accuratamente progettate e programmate dagli informatici. Ma è possibile che un agente possa *imparare* a cercare meglio? La risposta è sì, e il metodo si basa su un importante concetto, lo **spazio degli stati di metalivello**. Ogni stato in questo spazio rappresenta lo stato interno (computazionale) di un programma che sta eseguendo una ricerca in uno spazio degli stati ordinario, come può essere la mappa della Romania (per mantenere separati i due concetti, chiameremo la mappa della Romania **spazio degli stati a livello degli oggetti**). Per esempio, lo stato interno dell'algoritmo A* consiste nell'albero di ricerca corrente. Ogni

spazio degli stati a livello degli oggetti

azione nello spazio degli stati di metalivello è un passo computazionale che altera lo stato interno; per esempio, ogni passo computazionale in A^* espande un nodo foglia e aggiunge all’albero i suoi successori. Quindi la Figura 3.18, che mostra una sequenza di alberi di ricerca sempre più grandi, può essere vista come un cammino nello spazio degli stati di metalivello in cui ogni stato corrisponde a un albero di ricerca al livello degli oggetti.

Il cammino nella Figura 3.18 ha cinque passi tra cui uno, l’espansione di Fagaras, non particolarmente utile. Nei problemi più difficili si verificheranno molti passi falsi come quello, e un algoritmo di **apprendimento di metalivello** è in grado di imparare da tali esperienze per evitare di esplorare sottoalberi poco promettenti. Le tecniche usate per questo tipo di apprendimento sono descritte nel Capitolo 22 del Volume 2. Lo scopo dell’apprendimento è minimizzare il **costo totale** della risoluzione del problema, trovando il giusto equilibrio tra costo computazionale e costo di cammino.

apprendimento
di metalivello

3.6.6 Apprendere euristiche dall’esperienza

Abbiamo visto che un modo per trovare un’euristica è quello di immaginare un problema rilassato per cui sia facile trovare una soluzione ottima. Un’alternativa è quella di imparare dall’esperienza, che in questo caso può significare risolvere parecchi rompicapi a 8 tasselli, per esempio. Ogni soluzione ottima per un’istanza del rompicapo fornisce una coppia di esempio (obiettivo, cammino), e da questi esempi è possibile usare un algoritmo di apprendimento per costruire una funzione h che possa, con un po’ di fortuna, approssimare il costo di cammino reale per altri stati che dovessero presentarsi durante la ricerca. La maggior parte di questi approcci prevedono l’apprendimento di un’approssimazione imperfetta della funzione euristica, e quindi rischiano l’inammissibilità. Le tecniche per l’apprendimento automatico sono illustrate nel Capitolo 19 del Volume 2. Alla ricerca si possono applicare anche i metodi di apprendimento per rinforzo descritti nel Capitolo 22 del Volume 2.

Alcune tecniche di apprendimento automatico funzionano meglio quando si forniscono loro le **caratteristiche** (*feature*) di uno stato rilevanti per predire il valore dell’euristica di quello stato, invece della descrizione completa dello stato. Per esempio, la caratteristica “numero di tasselli fuori posto” potrebbe essere utile nella predizione dell’effettiva distanza di uno stato dall’obiettivo. Chiamiamo $x_1(n)$ questa caratteristica. Potremmo prendere 100 configurazioni generate casualmente di rompicapi a 8 tasselli e raccogliere statistiche sui costi effettivi delle loro soluzioni. Potremmo trovare che quando $x_1(n)$ vale 5, il costo medio della soluzione è intorno a 14, e così via. Naturalmente, possiamo sfruttare più di una caratteristica: una seconda caratteristica $x_2(n)$ potrebbe essere “il numero di coppie di tasselli adiacenti che sono adiacenti anche nello stato obiettivo”. Come dovrebbero essere combinate $x_1(n)$ e $x_2(n)$ per predire $h(n)$? Un approccio comune è usare una combinazione lineare:

$$h(n) = c_1x_1(n) + c_2x_2(n).$$

caratteristiche

Le costanti c_1 e c_2 sono calcolate per fornire il migliore adattamento ai dati raccolti su tutte le configurazioni generate casualmente. Ci si aspetta che c_1 e c_2 siano entrambe positive perché tasselli al posto sbagliato e coppie adiacenti sbagliate rendono il problema più difficile da risolvere. Notate che questa euristica soddisfa la condizione che $h(n) = 0$ per stati obiettivo, ma non è necessariamente ammissibile o consistente.

3.7 Riepilogo

In questo capitolo abbiamo presentato gli algoritmi di ricerca che un agente può utilizzare per scegliere sequenze di azioni all’interno di un’ampia varietà di ambienti, purché siano episodici, ad agente singolo, completamente osservabili, deterministici, statici e interamente noti. È necessario bilanciare il tempo necessario per la ricerca, la memoria disponibile e la

qualità della soluzione. Possiamo aumentare l'efficienza se disponiamo di conoscenza dipendente dal dominio sotto forma di una funzione euristica che possa stimare la lontananza di un dato stato dall'obiettivo, o se precalcoliamo soluzioni parziali utilizzando pattern o punti di riferimento.

- Prima che un agente possa cominciare a cercare, è necessario formulare un **problema** ben definito.
- Un problema è composto da cinque parti: lo **stato iniziale**, un insieme di **azioni**, un **modello di transizione** che descrive i risultati di tali azioni, un insieme di **stati obiettivo** e una funzione **costo di azione**.
- L'ambiente del problema è rappresentato dallo **spazio degli stati**. Un **cammino** attraverso lo spazio degli stati che porta dallo stato iniziale a uno stato obiettivo è una **soluzione**.
- Gli algoritmi di ricerca generalmente considerano stati e azioni come se fossero **atomici**, cioè privi di struttura interna (anche se abbiamo introdotto alcune caratteristiche degli stati nei paragrafi dedicati all'apprendimento).
- Gli algoritmi di ricerca sono valutati sulla base della loro **completezza**, **ottimalità**, **complessità temporale** e **complessità spaziale**.
- I metodi di **ricerca non informata** hanno accesso soltanto alla definizione del problema. Gli algoritmi costruiscono un albero di ricerca nel tentativo di trovare una soluzione, e differiscono tra loro in base a quale nodo espandono prima:
 - La **ricerca best-first** seleziona i nodi per l'espansione usando una **funzione di valutazione**.
 - La **ricerca in ampiezza** espande prima i nodi più vicini alla radice; è completa, ottima quando le azioni hanno costo unitario, ma ha una complessità spaziale esponenziale.
 - La **ricerca a costo uniforme** espande il nodo che ha minor costo di cammino, $g(n)$, ed è ottima per costi di passo qualsiasi.
 - La **ricerca in profondità** espande prima il nodo più profondo non ancora espanso. Non è completa né ottima, ma ha complessità spaziale lineare. La **ricerca a profondità limitata** prevede un limite alla profondità.
 - La **ricerca ad approfondimento iterativo** esegue una serie di ricerche a profondità limitata, estendendo progressivamente il limite di profondità finché non trova una soluzione. È completa quando viene effettuato un controllo completo di tutti i cicli, ottima quando le azioni hanno costo unitario, ha complessità temporale comparabile alla ricerca in ampiezza e complessità spaziale lineare.
 - La **ricerca bidirezionale** espande due frontiere, una attorno allo stato iniziale e una attorno all'obiettivo, fermandosi quando le due frontiere entrano in contatto.
- I metodi di **ricerca informata** possono avere accesso a una funzione **euristica** $h(n)$ che stima il costo di una soluzione a partire da n . Possono anche accedere a informazioni aggiuntive come database di pattern con i costi delle soluzioni.
 - La **ricerca best-first greedy** espande i nodi con $h(n)$ minima. Non è ottima, ma è spesso efficiente.
 - La **ricerca A*** espande i nodi con $f(n) = g(n) + h(n)$ minima. A* è completa e ottima, purché $h(n)$ sia ammissibile. La complessità spaziale di A* rappresenta ancora una difficoltà per molti problemi.
 - La ricerca **IDA*** (*Iterative Deepening A**) è una versione ad approfondimento iterativo di A* e ne affronta il problema della complessità spaziale.
 - **RBFS** (*Recursive Best-First Search*) e **SMA*** (*Simplified Memory-bounded A**) sono algoritmi robusti e ottimi che utilizzano una quantità di memoria limitata; dato un tempo sufficiente, possono risolvere problemi che A* non consente di risolvere perché esaurirebbe la memoria.

- La **ricerca beam** pone un limite alla dimensione della frontiera; ciò la rende incompleta e subottima, ma spesso riesce a trovare soluzioni ragionevolmente buone e viene eseguita più velocemente delle ricerche complete.
- La ricerca **A* pesata** focalizza la ricerca verso un obiettivo, espande meno nodi ma sacrifica l'ottimalità.
- Le prestazioni degli algoritmi di ricerca euristici dipendono dalla qualità della funzione euristica. Talvolta si possono costruire buone euristiche rilassando la definizione del problema, memorizzando in un database di pattern i costi precalcolati delle soluzioni, definendo dei punti di riferimento o apprendendo dall'esperienza.

ricerca beam

Note storiche e bibliografiche

Il tema della ricerca nello spazio degli stati risale ai primi anni dell'IA. Il lavoro di Newell e Simon su Logic Theorist (1957) e GPS (1961) portò negli anni 1960 a considerare gli algoritmi di ricerca come lo strumento principale nell'arsenale dei ricercatori di IA, e la risoluzione di problemi come attività precipua. Il lavoro nella ricerca operativa di Richard Bellman (1957) mostrò l'importanza dei costi di cammino additivi nella semplificazione degli algoritmi di ottimizzazione. Il testo di Nils Nilsson (1971) pose solide fondamenta teoriche per la disciplina.

Il rompicapo a 8 tasselli è il fratello minore di quello a 15 tasselli (gioco del 15), la cui storia è narrata dettagliatamente da Slocum e Sonneveld (2006). Nel 1880 il rompicapo a 15 tasselli ottenne l'attenzione del pubblico e dei matematici (Johnson e Story, 1879; Tait, 1880). I curatori dell'*American Journal of Mathematics* scrissero: "Il rompicapo a 15 tasselli, nelle ultime settimane, ha goduto di una posizione premiante nell'attenzione del pubblico americano, e si può affermare con certezza che ha interessato nove persone su dieci della comunità, di entrambi i sessi e di tutte le età e le condizioni sociali", mentre il *Weekly News-Democrat* di Emporia, in Kansas, scrisse il 12 marzo 1880 che: "È diventato una vera epidemia in tutto il paese".

Il famoso inventore di giochi americano Sam Loyd affermò falsamente di aver inventato il rompicapo a 15 tasselli (Loyd, 1959), ma in realtà il gioco fu inventato da Noyes Chapman, un postino di Canastota, New York, a metà degli anni 1870 (anche se un generico brevetto che copriva i giochi a tasselli scorrevoli fu concesso a Ernest Kinsey nel 1878). Ratner e Warmuth (1986) mostrarono che la versione generale $n \times n$ del rompicapo a 15 tasselli appartiene alla classe di problemi NP-completi.

Il cubo di Rubik fu inventato nel 1974 da Ernő Rubik, che scoprì anche un algoritmo per trovare soluzioni buone, ma non ottime. Korf (1997) trovò soluzioni ottime per alcune istanze di problemi casuali usando database di pattern e ricerca IDA*. Rokicki *et al.* (2014) dimostrarono che qualsiasi istanza può essere risolta in 26 mosse (se si considera una rotazione di 180° come due mosse, altrimenti in 20 mosse). La dimostrazione ha richiesto un tempo di calcolo pari a 35 anni di CPU; ma non conduce immediatamente a un algoritmo efficiente. Agostinelli *et al.* (2019) usaroni apprendimento con rinforzo, reti di deep learning e ricerche ad albero Monte Carlo per apprendere un risolutore del cubo di Rubik molto più efficiente. Non vi è garanzia di trovare una soluzione ottima rispetto al costo, ma la si ottiene nel 60% dei casi, e generalmente in meno di un secondo.

Ognuno dei problemi di ricerca nel mondo reale presentati in questo capitolo è stato oggetto di un grande sforzo di studio. I metodi per selezionare i voli di linea e generare pacchetti di viaggio in modo ottimale rimangono in gran parte proprietari, ma Carl de Marcken ha dimostrato mediante una riduzione ai problemi decisionali diofanteei che le tariffe delle compagnie aeree e le restrizioni imposte sono diventate così complesse e involte che scegliere un volo ottimale è oggi formalmente *indecidibile* (Robson, 2002). Il problema del commesso viaggiatore (TSP) è un classico problema combinatorio dell'informatica teorica (Lawler *et al.*, 1992). Karp (1972) ha dimostrato che il problema decisionale TSP è NP-difficile, ma per trattarlo sono stati sviluppati diversi metodi euristici approssimati (Lin e Kernighan, 1973). Arora (1998) ha definito uno schema di approssimazione completamente polinomiale per i problemi TSP euclidei. LePaugh (2010) ha fornito una rassegna dei metodi di

configurazione VLSI, e le riviste specializzate includono molti articoli sull'argomento. La navigazione dei robot è trattata nel Capitolo 26 del Volume 2. La prima dimostrazione di generazione di sequenze di montaggio automatico fu svolta da FREDDY (Michie, 1972); una rassegna completa è fornita da (Bahubalendruni e Biswal, 2016)

Gli algoritmi di ricerca non informata sono un argomento centrale dell'informatica (Carmen *et al.*, 2009) e della ricerca operativa (Dreyfus, 1969). La ricerca in ampiezza fu formulata da Moore (1959) per risolvere labirinti. Il metodo della programmazione dinamica (Bellman, 1957; Bellman e Dreyfus, 1962), che registra sistematicamente le soluzioni di tutti i sottoproblemi di lunghezza crescente, può essere considerato una forma di ricerca in ampiezza.

L'algoritmo di Dijkstra nella forma in cui è solitamente presentato (Dijkstra, 1959) è applicabile a grafi finiti espliciti. Nilsson (1971) introdusse una versione che chiamò ricerca a costo uniforme (perché l'algoritmo “si espande lungo confini di pari costo di cammino”) che si applica a grafi infiniti definiti implicitamente. Il lavoro di Nilsson introdusse anche il concetto di liste chiuse e aperte e il termine “ricerca su grafo”. Il termine “ricerca best-first” fu introdotto in *Handbook of AI* (Barr e Feigenbaum, 1981). Gli algoritmi di Floyd-Warshall (Floyd, 1962) e Bellman-Ford (Bellman, 1958; Ford, 1956) consentono costi di azione negativi (purché non vi siano cicli negativi).

Una versione dell'approfondimento iterativo progettata per utilizzare al meglio il tempo a disposizione in una partita di scacchi fu usata per la prima volta da Slate e Atkin (1977) nel programma scacchistico CHESS 4.5. L'algoritmo B di Martelli (1977) include anche un aspetto di approfondimento iterativo. La tecnica di approfondimento iterativo fu introdotta da Bertram Raphael (1976) e portata all'attenzione generale da Korf (1985a).

L'uso di informazione euristica per la risoluzione di problemi compare in uno dei primi articoli di Simon e Newell (1958), ma il termine “ricerca euristica” e l'uso di funzioni euristiche che stimano la distanza dall'obiettivo sono di poco successivi (Newell ed Ernst, 1965; Lin, 1965). Doran e Michie (1966) svolsero estesi studi sperimentali sulla ricerca euristica. Benché abbiano analizzato la lunghezza dei cammini e la “penetrazione” (il rapporto tra la lunghezza del cammino e il numero totale di nodi esaminati), sembra che abbiano ignorato l'informazione fornita dal costo del cammino $g(n)$. L'algoritmo A*, che incorpora il costo del cammino corrente nella ricerca euristi-

ca, fu sviluppato da Hart, Nilsson e Raphael (1968). Dechter e Pearls (1985) studiarono le condizioni sotto le quali A* è ottimamente efficiente (relativamente al numero di nodi espansi).

L'articolo originale su A* (Hart *et al.*, 1968) introduce il requisito della consistenza delle funzioni euristiche. Il requisito di monotonicità fu introdotto da Pohl (1977) come sostituto più semplice del precedente, ma Pearl (1984) ha dimostrato che i due sono equivalenti.

Pohl (1977) fu il primo a studiare la relazione tra gli errori nelle funzioni euristiche e la complessità temporale di A*. Risultati di base furono ottenuti per la ricerca ad albero con costi di azione unitari e un singolo stato obiettivo (Pohl, 1977; Gaschnig, 1979; Huyn *et al.*, 1980; Pearl, 1984) e con stati obiettivo multipli (Dinh *et al.*, 2007). Korf e Reid (1998) mostraron come predire il numero esatto di nodi espansi (non solo un'approssimazione asintotica) su una varietà di domini di problemi reali. Il “fattore di ramificazione effettivo” fu proposto da Nilsson (1971) come misura dell'efficienza della ricerca euristica. Per la ricerca su grafo, Helmert e Röger (2008) notarono che diversi problemi ben noti contenevano un numero esponenziale di nodi su cammini che rappresentano soluzioni ottime rispetto al costo, il che implica una complessità temporale esponenziale per A*.

Esistono molte varianti dell'algoritmo A*. Pohl (1970) introdusse la ricerca A* pesata, e in seguito una versione dinamica (1973) dove i pesi cambiano in base alla profondità nell'albero. Ebendt e Drechsler (2009) fecero una sintesi dei risultati ed esamarono alcune applicazioni. Hatem e Ruml (2014) presentarono una versione semplificata e migliorata dell'algoritmo A* pesato, più facile da implementare. Wilt e Ruml (2014) introdussero come alternativa alla ricerca *greedy* la ricerca *speedy*, focalizzata sul minimizzare il tempo di ricerca, e Wilt e Ruml (2016) mostraron che le migliori euristiche per ricerche soddisfacenti sono diverse da quelle per ricerche ottimali. Burns *et al.* (2012) fornirono alcune tecniche di implementazione per scrivere codice per ricerche veloci, e Felner (2018) considerò come l'implementazione cambi quando si utilizza un test obiettivo anticipato.

Pohl (1971) introdusse la ricerca bidirezionale. Holte *et al.* (2016) descrisse la versione della ricerca bidirezionale che garantisce l'incontro a metà strada, cosa che ne amplia il campo di applicazione. Eckerle *et al.* (2017) descrissero l'insieme delle coppie di nodi sicuramente espansi e mostraron che nessuna ricerca bidirezionale può essere ottimamente efficiente. L'al-

goritmo NBS (Chen *et al.*, 2017) fa uso esplicito di una coda di coppie di nodi.

Una combinazione di A* bidirezionale e punti di riferimento noti è stata utilizzata per trovare in modo efficiente itinerari stradali per il servizio di mappe online di Microsoft (Goldberg *et al.*, 2006). Dopo aver memorizzato in una cache un insieme di cammini tra punti di riferimento, l'algoritmo è in grado di trovare un cammino ottimo rispetto al costo tra qualsiasi coppia di punti in un grafo con 24 milioni di punti che rappresenta gli Stati Uniti, ricercando meno dello 0,1% del grafo. Korf (1987) mostrò come utilizzare subbiettivi, macro-operatori e le astrazioni per ottenere notevoli incrementi di velocità rispetto alle tecniche precedenti. Delling *et al.* (2009) descrissero come utilizzare ricerca bidirezionale, punti di riferimento, struttura gerarchica e altre tecniche per trovare itinerari stradali. Anderson *et al.* (2008) descrissero una tecnica correlata, chiamata **ricerca ad affinamento (coarse-to-fine)**, che può essere pensata come la definizione di punti di riferimento a vari livelli gerarchici di astrazione. Korf (1987) descrisse le condizioni sotto le quali la ricerca ad affinamento fornisce un aumento di velocità esponenziale. Knoblock (1991) fornì risultati sperimentali e analisi per quantificare i vantaggi della ricerca gerarchica.

A* e altri algoritmi di ricerca nello spazio degli stati sono strettamente imparentati con le tecniche di **branch-and-bound** ampiamente diffuse nel campo della ricerca operativa (Lawler e Wood, 1966; Rayward-Smith *et al.*, 1966). Kumar e Kanal (1988) hanno tentato una “grande unificazione” di ricerca euristica, programmazione dinamica e tecniche branch-and-bound sotto il nome di CDP (acronimo di *Composite Decision Process*).

Dato che i computer negli anni 1960 avevano poche migliaia di parole di memoria, non stupisce che la ricerca euristica a memoria limitata sia stata subito oggetto di ricerca. Il Graph Traverser (Doran e Michie, 1966), uno dei primi programmi di ricerca, sceglie un’azione dopo aver svolto una ricerca best-first fino al limite della memoria disponibile. IDA* (Korf, 1985b) è stato il primo algoritmo di ricerca euristica ottimale rispetto alla lunghezza, a memoria limitata e di largo uso, e ne sono state sviluppate molte varianti. Un’analisi dell’efficienza di IDA* e dei suoi problemi con euristiche a valori reali è riportata in Patrick *et al.* (1992).

La versione originale di RBFS (Korf, 1993) è in realtà un po’ più complicata dell’algoritmo mostrato nella Figura 3.22, che assomiglia di più a un algoritmo

sviluppato indipendentemente e chiamato **espansione iterativa**, o IE (Russell, 1992). RBFS usa un limite inferiore oltre che superiore; i due algoritmi si comportano in maniera identica con euristiche ammissibili, ma RBFS espande i nodi in ordine best-first anche quando l’euristica non è ammissibile. L’idea di tener traccia del miglior cammino alternativo è apparsa per la prima volta nell’elegante implementazione in Prolog di A* da parte di Bratko (2009) e nell’algoritmo DTA* (Russell e Wefald, 1991). Quest’ultimo lavoro discute anche gli spazi degli stati di metalivello e l’apprendimento di metalivello.

L’algoritmo MA* apparve in Chakrabarti *et al.* (1989). SMA*, o Simplified MA*, emerse da un tentativo di implementare MA* (Russell, 1992). Kaindl e Khorsand (1994) hanno applicato SMA* per realizzare un algoritmo di ricerca bidirezionale molto più veloce dei precedenti. Korf e Zhang (2000) descrivono un approccio *divide et impera*, mentre Zhou e Hansen (2002) hanno introdotto la ricerca A* a memoria limitata su grafo e una strategia per passare a una ricerca in ampiezza per migliorare l’efficienza nell’uso della memoria (Zhou e Hansen, 2006).

L’idea che si possano derivare euristiche ammissibili dal rilassamento dei problemi è apparsa in un fondamentale articolo di Held e Karp (1970), che hanno usato l’euristica del minimo albero di copertura (*minimum spanning tree*) per risolvere il problema del commesso viaggiatore (cfr. Esercizio 3.MSTR). L’automazione del processo di rilassamento è stata implementata con successo da Prieditis (1993). Vi è ampia e crescente letteratura sull’applicazione dell’apprendimento automatico alla scoperta di funzioni euristiche (Samadi *et al.*, 2008; Arfaee *et al.*, 2010; Thayer *et al.*, 2011; Lelis *et al.*, 2012).

L’uso di database di pattern per derivare euristiche ammissibili è dovuto a Gasser (1995) e Culberson e Schaeffer (1996, 1998); i database di pattern disgiunti sono descritti da Korf e Felner (2002). L’interpretazione probabilistica delle euristiche è stata studiata in profondità da Pearl (1984) e Hansson e Mayer (1989).

Tra i libri più influenti nel campo degli algoritmi di ricerca vi sono *Heuristics* di Pearl (1984) e *Heuristic Search* di Edelkamp e Schrödl (2012). Articoli sui nuovi algoritmi di ricerca sono presentati all’International Symposium on Combinatorial Search (SoCS), all’International Conference on Automated Planning and Scheduling (ICAPS) e nelle conferenze generali sull’IA come AAAI e IJCAI, oltre che pubblicati su riviste quali *Artificial Intelligence* e *Journal of the ACM*.

Ricerca in ambienti complessi

- 4.1 Ricerca locale e problemi di ottimizzazione
- 4.2 Ricerca locale in spazi continui
- 4.3 Ricerca con azioni non deterministiche
- 4.4 Ricerca con osservazioni parziali
- 4.5 Agenti per ricerca online e ambienti sconosciuti
- 4.6 Riepilogo
Note storiche e bibliografiche

In cui rilassiamo le ipotesi semplificative del capitolo precedente, avvicinandoci così al mondo reale.

Nel Capitolo 3 abbiamo trattato problemi in ambienti completamente osservabili, deterministici, statici, noti in cui la soluzione è una sequenza di azioni. In questo capitolo rilassiamo questi vincoli. Iniziamo con il problema di trovare un buono stato senza preoccuparci del cammino per raggiungerlo, considerando sia stati discreti (Paragrafo 4.1) sia continui (Paragrafo 4.2). Poi rilassiamo le ipotesi di determinismo (Paragrafo 4.3) e osservabilità (Paragrafo 4.4). In un mondo non deterministico, l'agente avrà bisogno di un piano condizionale e di eseguire azioni diverse a seconda di ciò che osserva, per esempio fermandosi al semaforo se è rosso e proseguendo se è verde. In caso di osservabilità parziale, l'agente dovrà anche tenere traccia degli stati in cui potrebbe trovarsi. Infine, nel Paragrafo 4.5 vediamo come guidare l'agente attraverso uno spazio sconosciuto che deve apprendere mentre procede, usando la **ricerca online**.

4.1 Ricerca locale e problemi di ottimizzazione

Nei problemi di ricerca del Capitolo 3 volevamo trovare cammini attraverso lo spazio degli stati, per esempio da Arad a Bucarest. A volte invece siamo interessati soltanto allo stato finale, non al cammino percorso per raggiungerlo. Per esempio, nel problema delle 8 regine (Figura 4.3) ci interessa soltanto trovare una configurazione valida delle 8 regine (infatti, conoscendo la configurazione, è facile ricostruire i passi per crearla). Questo vale anche per molte applicazioni importanti come la progettazione di circuiti integrati, il layout di impianti industriali, il *job-shop scheduling* (assegnamento nel tempo di macchine a compiti), la programmazione automatica, l'ottimizzazione di reti di telecomunicazioni, la programmazione delle colture e la gestione di portafogli di azioni.

ricerca locale

Gli algoritmi di **ricerca locale** operano cercando a partire da uno stato iniziale e procedendo verso gli stati adiacenti senza tenere traccia dei cammini, né degli stati già raggiunti. Questo significa che non sono sistematici – potrebbero anche non esplorare mai una porzione dello spazio degli stati dove in effetti risiede una soluzione. Hanno però due vantaggi: (1) usano pochissima memoria e (2) spesso riescono a trovare soluzioni ragionevoli in spazi degli stati grandi o anche infiniti per cui usare algoritmi sistematici non sarebbe praticabile.

Gli algoritmi di ricerca locale possono anche risolvere **problematiche di ottimizzazione**, in cui lo scopo è trovare lo stato migliore secondo una **funzione obiettivo**.

Per comprendere la ricerca locale, potete considerare gli stati di un problema disposti in un **panorama dello spazio degli stati** (*state space landscape*), come nella Figura 4.1. Ogni punto (stato) in nel panorama ha una “altezza” definita dal valore della funzione obiettivo; se l’altezza corrisponde a una funzione obiettivo, lo scopo è quello di trovare il picco più alto – un **massimo globale** – e chiamiamo questo processo **hill climbing** (traducibile letteralmente in “scalare la collina”). Se l’altezza corrisponde al costo, lo scopo è quello di trovare l’avallamento più profondo – un **minimo globale** – e parliamo in questo caso di **discesa del gradiente** (*gradient descent*).

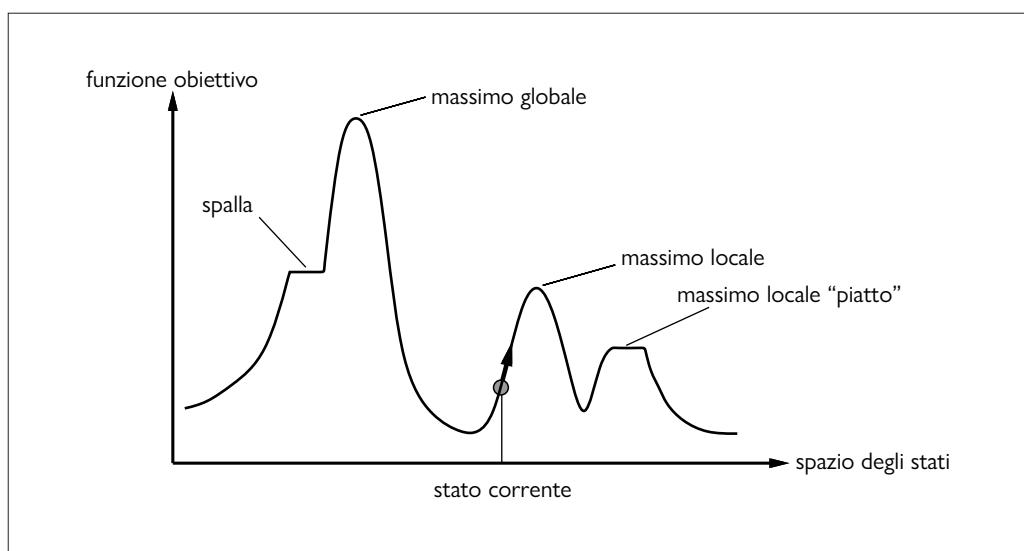
4.1.1 Ricerca hill climbing

hill climbing

L’algoritmo di ricerca **hill climbing** è mostrato nella Figura 4.2; tiene traccia di un solo stato corrente e a ogni iterazione passa allo stato vicino con valore più alto, cioè punta nella direzione che presenta l’**ascesa più ripida** (*steepest ascent*), senza guardare oltre gli stati immediati.

Figura 4.1

Un panorama dello spazio degli stati monodimensionale, in cui l’altezza corrisponde alla funzione obiettivo. Lo scopo è trovare il massimo globale.



```

function HILL-CLIMBING(problema) returns uno stato che è un massimo locale
  corrente  $\leftarrow$  problema.STATOINIZIALE
  while true do
    vicino  $\leftarrow$  lo stato successore di corrente di valore più alto
    if VALORE(vicino)  $\leq$  VALORE(corrente) then return corrente
    corrente  $\leftarrow$  vicino

```

Figura 4.2 L'algoritmo di ricerca hill climbing, che rappresenta la tecnica più semplice di ricerca locale. A ogni passo il nodo corrente viene rimpiazzato dal vicino migliore.

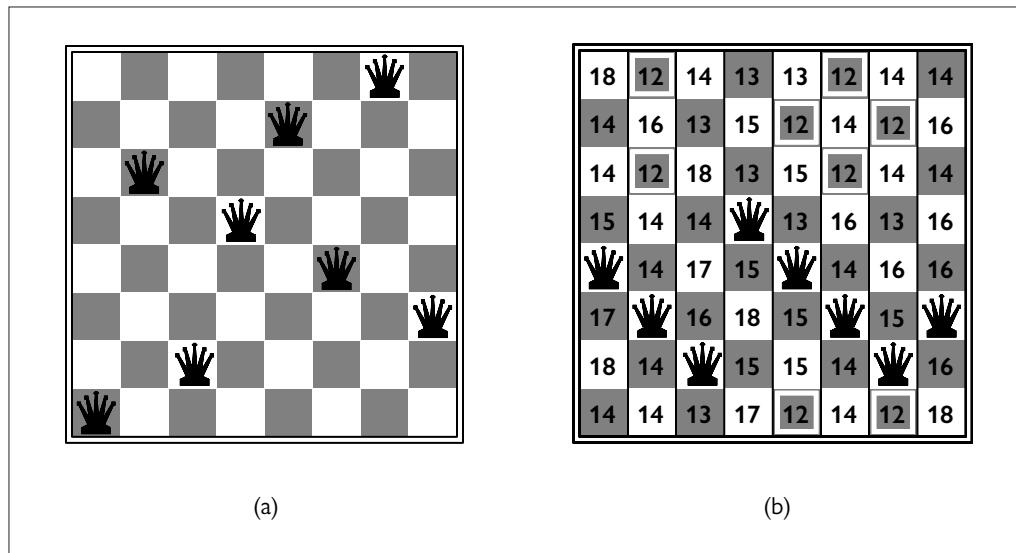


Figura 4.3 (a) Il problema delle 8 regine: posizionate 8 regine su una scacchiera in modo che nessuna ne attacchi un'altra (una regina attacca ogni altro pezzo nella stessa riga, colonna o diagonale). Questa posizione è quasi una soluzione, eccetto per le due regine nella quarta e settima colonna che si attaccano tra loro lungo la diagonale. (b) Uno stato del problema delle 8 regine con stima euristica di costo $h = 17$. La scacchiera mostra il valore di h per ogni possibile successore ottenuto muovendo una regina nella sua colonna. Ci sono 8 mosse migliori a pari merito, con $h = 12$; l'algoritmo hill climbing sceglierà una di queste.

tamente vicini a quello corrente. Assomiglia un po' al tentativo di trovare la cima dell'Everest in condizioni di forte nebbia mentre si soffre di amnesia. Notate che un modo di usare questo algoritmo consiste nell'usare come funzione obiettivo il valore negativo di una funzione di costo euristica; così si sale localmente allo stato con la minima distanza euristica dall'obiettivo.

Per illustrare l'algoritmo hill climbing utilizzeremo il **problema delle 8 regine** (Figura 4.3) con una **formulazione a stato completo**, in cui ogni stato ha tutti i componenti di una soluzione, che però potrebbero non essere tutti al posto giusto. In questo caso ogni stato ha 8 regine sulla scacchiera, una per colonna. Lo stato iniziale è scelto a caso, e i successori di uno stato sono tutti i possibili stati generati muovendo una singola regina in un'altra casella nella stessa colonna (perciò ogni stato ha $8 \times 7 = 56$ successori). La funzione di costo euristica h è il numero di coppie di regine che si attaccano, e sarà zero soltanto per le soluzioni (si conta un attacco se due pezzi sono allineati, anche se vi è un altro pezzo che si frappone tra di essi). La Figura 4.3(b) mostra uno stato in cui $h = 17$. La figura mostra i valori di h di tutti i successori.

formulazione a
stato completo

L'algoritmo hill climbing viene talvolta chiamato **ricerca locale greedy**, perché sceglie uno stato vicino “buono” senza pensare a come andrà avanti. Benché l'avarizia sia considerata uno

ricerca locale
greedy

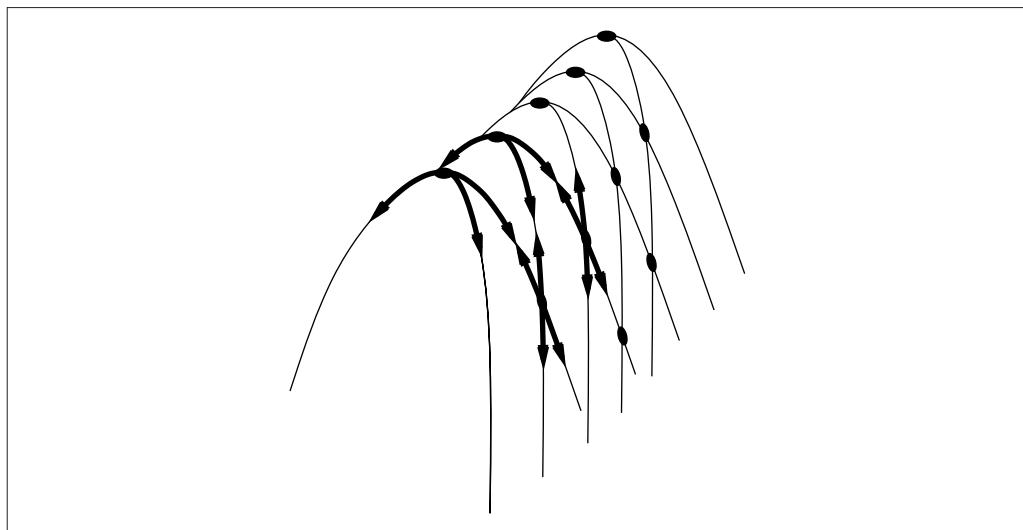


Figura 4.4 Il disegno mostra perché le creste causano difficoltà agli algoritmi di hill climbing. La griglia di stati (piccoli ovali scuri) è sovrapposta a una cresta che sale da sinistra a destra, creando una sequenza di massimi locali che non sono direttamente collegati l'uno all'altro. Da ogni massimo locale, tutte le azioni possibili puntano “verso valle”. Topologie come questa sono comuni negli spazi degli stati a basso numero di dimensioni, come i punti in un piano bidimensionale. In spazi degli stati con centinaia o migliaia di dimensioni, invece, questa immagine intuitiva non vale e solitamente esistono almeno alcune dimensioni che consentono di evitare creste e plateau.

dei sette vizi capitali, gli algoritmi “avari” (*greedy*) si comportano spesso molto bene. L’hill climbing può procedere molto rapidamente verso la soluzione, perché di solito è abbastanza facile migliorare uno stato sfavorevole. Per esempio, dallo stato della Figura 4.3(b) servono solo cinque passi per raggiungere quello della Figura 4.3(a), che ha $h = 1$ ed è vicinissimo a una soluzione. Sfortunatamente, spesso l’hill climbing rimane bloccato per le seguenti ragioni:

massimo locale

- **massimi locali:** un massimo locale è un picco più alto degli stati vicini, ma inferiore al massimo globale. Gli algoritmi hill climbing che raggiungono la vicinanza di un massimo locale saranno attratti verso il picco, ma rimarranno bloccati lì senza poter andare altrove. La Figura 4.1 illustra schematicamente il problema. Più concretamente, lo stato nella Figura 4.3(a) è un massimo locale (cioè, un minimo locale della funzione di costo h); ogni mossa di una singola regina peggiora la situazione;

cresta

- **creste:** una cresta (*ridge*) è mostrata nella Figura 4.4. Le creste danno origine a una sequenza di massimi locali molto difficili da esplorare da parte degli algoritmi greedy;

plateau spalla

- **plateau:** un plateau è un’area piatta del panorama dello spazio degli stati. Può essere un massimo locale piatto, da cui non è possibile fare ulteriori progressi, oppure una **spalla** (*shoulder*), da cui si potrà salire ulteriormente (Figura 4.1). Una ricerca hill climbing potrebbe perdere sul plateau.

In ognuno di questi casi, l’algoritmo raggiunge un punto dal quale non riesce a compiere ulteriori progressi. Partendo da uno stato generato casualmente del problema delle 8 regine, l’algoritmo hill climbing si blocca l’86% delle volte, risolvendo solo il 14% delle istanze. Funziona molto velocemente, richiedendo in media solo 4 passi per trovare una soluzione e 3 quando si blocca: non male, per uno spazio degli stati con $8^8 \approx 17$ milioni di stati.

mossa laterale

Come possiamo risolvere un maggior numero di istanze? Un modo è quello di proseguire nella ricerca una volta raggiunto un plateau, consentendo una **mossa laterale** nella speranza che il plateau sia in realtà una spalla, come si vede nella Figura 4.1. Ma se fossimo realmente su un massimo locale piatto, con questo approccio finiremmo per vagare sul plateau per sem-

pre. Possiamo quindi porre un limite al numero di mosse laterali consecutive, per esempio fermandoci dopo 100. Così si aumenta la percentuale di istanze di problemi risolte dall'algoritmo di hill climbing dal 14% al 94%. Il successo ha anche un costo: l'algoritmo esegue in media 21 passi per ogni successo e 64 per ogni fallimento.

Sono state inventate molte varianti dell'hill climbing. L'**hill climbing stocastico** sceglie a caso tra tutte le mosse che vanno verso l'alto: la probabilità della scelta può essere influenzata dalla "pendenza" delle mosse. Normalmente questo algoritmo converge più lentamente di quello che sceglie sempre la mossa più conveniente, ma in alcuni panorami di stati è capace di trovare soluzioni migliori. L'**hill climbing con prima scelta** implementa la precedente versione stocastica generando casualmente i successori fino a ottenerne uno preferibile allo stato corrente. Questa strategia è molto buona quando uno stato ha molti successori (per esempio migliaia).

Un'altra variante è l'**hill climbing con riavvio casuale**, che adotta il vecchio adagio: "Se all'inizio non ce la fai riprova, e poi riprova ancora". L'algoritmo conduce una serie di ricerche hill climbing partendo da stati iniziali generati casualmente, fino a quando raggiunge un obiettivo. È completo con probabilità 1, perché prima o poi dovrà generare, come stato iniziale, proprio un obiettivo. Se ogni ricerca hill climbing ha una probabilità p di successo, il numero atteso di riavvii richiesti è $1/p$. Per istanze del problema delle 8 regine senza mosse laterali, $p \approx 0,14$, quindi saranno necessarie circa 7 iterazioni per trovare uno stato obiettivo (6 fallimenti e 1 successo). Il numero atteso di passi è il costo dell'iterazione che ha successo più $(1-p)/p$ volte il costo del fallimento, o circa 22 passi in totale. Permettendo le mosse laterali, sono richieste in media $1/0,94 \approx 1,06$ iterazioni e $(1 \times 21) + (0,06/0,94) \times 64 \approx 25$ passi. Nel caso delle 8 regine, quindi, l'hill climbing con riavvio casuale è davvero molto efficace. Anche con tre milioni di regine, questo approccio può trovare soluzioni in tempi dell'ordine dei secondi.¹

Il successo dell'hill climbing dipende molto dalla forma del panorama dello spazio degli stati: se ci sono pochi massimi locali e plateau, quello con riavvio casuale troverà molto velocemente una buona soluzione. D'altra parte, molti problemi reali hanno un panorama che assomiglia a una famiglia di porcospini disseminata su un pavimento piatto, con piccoli porcospini appoggiati alla punta di ogni spina dei porcospini più grandi. I problemi NP-difficili (cfr. Appendice A) hanno tipicamente un numero esponenziale di massimi locali in cui ci si può incastrare. Nonostante questo, spesso è possibile trovare un massimo locale ragionevolmente buono con un numero abbastanza piccolo di riavvii.

4.1.2 Simulated annealing

Un algoritmo hill climbing che non scende mai "a valle" verso stati con valore più basso (o costo più alto) è sempre vulnerabile alla possibilità di rimanere bloccato in corrispondenza di un massimo locale. Di contro, un'esplorazione del tutto casuale che si muove verso uno stato successore senza preoccuparsi del valore finirà per incontrare il massimo globale, ma sarà estremamente inefficiente. Sembra ragionevole, quindi, cercare di combinare in qualche modo l'hill climbing con un'esplorazione casuale in modo da ottenere sia l'efficienza che la completezza.

Un algoritmo che fa proprio questo è il **simulated annealing** (letteralmente, "tempra simulata"). In metallurgia, l'**annealing** (tempra) è il processo usato per indurire i metalli o il vetro riscaldandoli ad altissime temperature e raffreddandoli gradualmente, permettendo così al materiale di cristallizzare in uno stato a bassa energia. Per spiegare il simulated annealing, cambiamo il punto di vista dalla scalata di colline alla **discesa del gradiente** (cioè la

**hill climbing
stocastico**

**hill climbing con
prima scelta**

**hill climbing con
riavvio casuale**

**simulated
annealing**

¹ Luby *et al.* (1993) suggeriscono di ripartire dopo un certo numero di passi e dimostrano che questo approccio può essere molto più efficiente di permettere a ogni ricerca di continuare indefinitamente.

```

function SIMULATED-ANNEALING(problema, velocità_raffreddamento) returns uno stato soluzione
  corrente  $\leftarrow$  problema.STATOINIZIALE
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  velocità_raffreddamento[t]
    if T = 0 then return corrente
    successivo  $\leftarrow$  un successore scelto a caso di corrente
     $\Delta E \leftarrow$  VALORE(corrente) – VALORE(successivo)
    if  $\Delta E > 0$  then corrente  $\leftarrow$  successivo
    else corrente  $\leftarrow$  successivo solo con probabilità  $e^{-\Delta E/T}$ 

```

Figura 4.5 L'algoritmo simulated annealing, una versione dell'hill climbing stocastico che permette di compiere alcune mosse “verso valle”. L'input *velocità_raffreddamento* determina il valore della “temperatura” *T* in funzione del tempo.

minimizzazione del costo) e immaginiamoci alle prese con il problema di far entrare una pallina da ping-pong nella fessura più profonda di una superficie molto corrugata. Se lasciamo semplicemente rotolare la pallina, questa si fermerà in corrispondenza di un minimo locale. Se scuotiamo la superficie, possiamo far uscire la pallina da questo avvallamento, rischiando però di farla entrare in un altro minimo più profondo, dove starà ancora per più tempo. Il trucco è scuotere abbastanza da spostare la pallina fuori dai minimi locali, ma non così tanto da farla uscire anche dal minimo globale. La soluzione proposta dal simulated annealing è di cominciare scuotendo molto (alta temperatura) e poi ridurre gradualmente l'intensità dello scuotimento (riducendo la temperatura).

La struttura complessiva dell'algoritmo simulated annealing (Figura 4.5) è simile a quella dell'hill climbing: stavolta però, invece della mossa *migliore*, viene scelta una mossa *casuale*. Se la mossa migliora la situazione, viene sempre accettata; in caso contrario l'algoritmo la accetta con una probabilità inferiore a 1. La probabilità decresce esponenzialmente con la “cattiva qualità” della mossa, misurata dal peggioramento ΔE della valutazione. La probabilità decresce anche con la “temperatura” *T* che scende costantemente: le mosse “cattive” saranno accettate più facilmente all'inizio, in condizioni di *T* alta, e diventeranno sempre meno probabili a mano a mano che *T* si abbassa. Se la *velocità_raffreddamento* fa decrescere la temperatura da *T* a 0 abbastanza lentamente, per una proprietà della distribuzione di Boltzmann, $e^{\Delta E/T}$, tutta la probabilità è concentrata sui massimi globali, che l'algoritmo troverà con probabilità tendente a 1.

Il simulated annealing è stato usato estensivamente per risolvere problemi di configurazione VLSI a partire dagli anni 1980. In seguito è stato applicato ampiamente nello scheduling delle fabbriche e in altri problemi di ottimizzazione su larga scala.

4.1.3 Ricerca local beam

ricerca local beam

Memorizzare un solo nodo può sembrare una reazione estrema ai problemi legati alle limitazioni di memoria. L'algoritmo di **ricerca local beam** tiene traccia di *k* stati anziché uno solo. All'inizio comincia con *k* stati generati casualmente: a ogni passo, sono generati i successori di tutti i *k* stati. Se uno qualsiasi di essi è un obiettivo, l'algoritmo termina; altrimenti sceglie i *k* successori migliori dalla lista di tutti i successori e ricomincia.

A prima vista, una ricerca local beam con *k* stati potrebbe sembrare nulla più di *k* ricerche a riavvio casuale eseguite in parallelo anziché una dopo l'altra. In realtà, i due algoritmi sono piuttosto differenti. In una ricerca a riavvio casuale, ogni processo di ricerca è del tutto indipendente dagli altri. In una ricerca local beam, informazione utile viene passata dall'uno all'altro *thread* di ricerca paralleli. In effetti, lo stato che genera i migliori successori dice agli



altri: “Venite qui, l’erba è più verde!”. L’algoritmo abbandona rapidamente le ricerche infruttuose e sposta la sue risorse dove si stanno verificando i progressi maggiori.

La ricerca local beam può soffrire di una carenza di diversificazione tra i k stati, che possono concentrarsi in cluster in una piccola regione dello spazio degli stati, rendendo così questa ricerca poco più di una versione k volte più lenta dell’hill climbing. Una variante chiamata **ricerca beam stocastica**, analoga all’hill climbing stocastico, aiuta ad alleviare questo problema: invece di scegliere i migliori k successori, in questo caso si scelgono i successori con probabilità proporzionale al loro valore, aumentando così la diversificazione.

**ricerca beam
stocastica**

4.1.4 Algoritmi evolutivi

Gli **algoritmi evolutivi** possono essere visti come varianti della ricerca beam stocastica esplcitamente motivati dalla metafora della selezione naturale in biologia: esiste una popolazione di individui (stati) in cui quelli più adatti (valore più alto) producono prole (stati successori) che forma la prossima generazione, in un processo chiamato **ricombinazione**. Esistono numerosissime forme di algoritmi evolutivi, che variano per i seguenti aspetti.

algoritmo evolutivo

- La dimensione della popolazione.
- La rappresentazione di ciascun individuo. Negli **algoritmi genetici** ogni individuo è una stringa su un alfabeto finito (spesso una stringa booleana), esattamente come il DNA è una stringa sull’alfabeto **ACGT**. Nelle **strategie evolutive**, un individuo è una sequenza di numeri reali. Nella **programmazione genetica**, un individuo è un programma per computer.
- Il numero di genitori, ρ , che si riuniscono a generare la prole. Il caso più comune è $\rho = 2$: due genitori combinano i loro “geni” (parti della loro rappresentazione) per generare la prole. Quando $\rho = 1$ abbiamo la ricerca beam stocastica (che può essere considerata come una riproduzione asessuata). È possibile avere $\rho > 2$, caso che si verifica raramente in natura ma è abbastanza facile da simulare al computer.
- Il processo di **selezione** per selezionare gli individui che diventeranno i genitori della generazione successiva: una possibilità è quella di selezionarli tra tutti gli individui con probabilità proporzionale al loro punteggio di adattamento (*fitness*), un’altra è quella di scegliere a caso n individui ($n > \rho$) e poi selezionare i ρ più adatti come genitori.
- La procedura di ricombinazione. Un approccio comune (assumendo $\rho = 2$) è quello di selezionare a caso un **punto di crossover** (incrocio) per suddividere ognuna delle stringhe genitori e poi ricombinare le parti per formare due figli: uno con la prima parte del genitore 1 e la seconde parte del genitore 2; l’altro con la seconda parte del genitore 1 e la prima parte del genitore 2.
- Il **tasso di mutazione**, che determina la frequenza con cui la prole presenta mutazioni casuali della loro rappresentazione. Una volta generata la prole, ogni bit che la compone viene invertito con probabilità uguale al tasso di mutazione.
- La formazione della nuova generazione. Può essere semplicemente la nuova prole formata, oppure può includere alcuni genitori con il migliore punteggio dalla generazione precedente (una prassi chiamata **elitismo** che garantisce che l’adattabilità totale non diminuirà mai nel tempo). La pratica dell’**abbattimento** (*culling*) in cui tutti gli individui al di sotto di una certa soglia sono scartati, può consentire di velocizzare l’esecuzione (Baum *et al.*, 1995).

algoritmo genetico

strategia evolutiva
**programmazione
genetica**

selezione

punto di crossover

tasso di mutazione

elitismo

La Figura 4.6(a) mostra una popolazione costituita da quattro stringhe di 8 cifre, ognuna delle quali rappresenta uno stato del rompicapo delle 8 regine: la cifra c -esima rappresenta il numero di riga della regina presente nella colonna c . In (b), ogni stato ha un valore dato dalla funzione di fitness. Valori della fitness più alti sono migliori, perciò nel problema delle 8 regine utilizziamo il numero di coppie di regine *che non si attaccano*, che vale $8 \times 7/2 = 28$ per una soluzione. I valori dei quattro stati in (b) sono 24, 23, 20 e 11. I punteggi della fitness

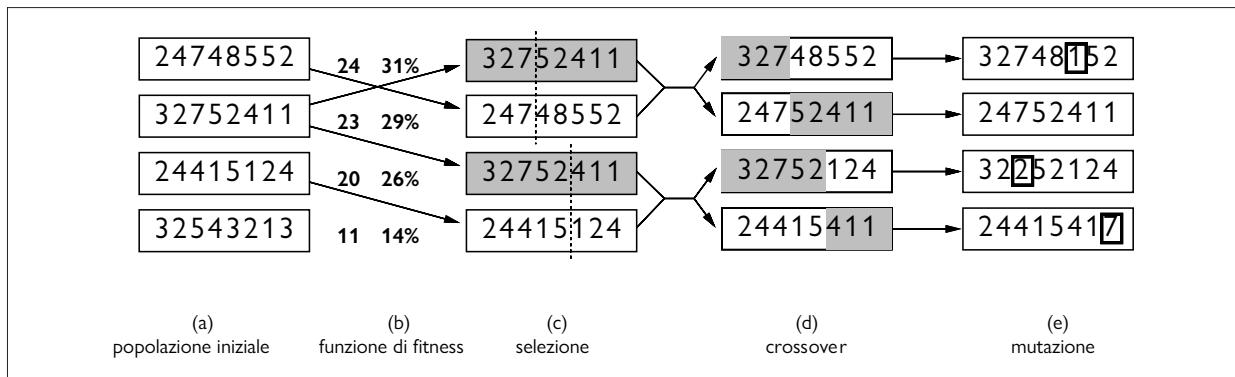


Figura 4.6 Un algoritmo genetico illustrato con stringhe di cifre che rappresentano gli stati del problema delle 8 regine. La popolazione iniziale in (a) è ordinata in base alla funzione di fitness in (b), dando come risultato le coppie di (c). Queste ultime danno origine alla prole in (d), che è poi soggetta a mutazione in (e).

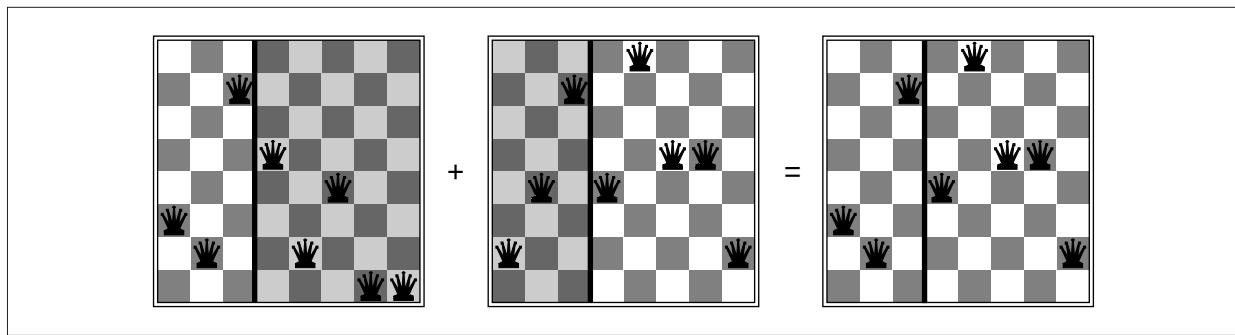


Figura 4.7 Gli stati del problema delle 8 regine corrispondenti ai primi due genitori della Figura 4.6(c) e al primo stato della prole della Figura 4.6(d). Le colonne ombreggiate sono perse nel passo di crossover, quelle chiare mantengono (per interpretare i numeri della Figura 4.6: la riga 1 è quella più in basso, la riga 8 è quella più in alto).

sono poi normalizzati come probabilità, e i valori risultanti sono mostrati accanto ai valori della fitness nella Figura 4.6(b).

In (c) vengono selezionate due coppie di genitori, in base alle probabilità della Figura 4.6(b). Per ogni coppia selezionata viene scelto casualmente un punto di crossover (linea tratteggiata). In (d) incrociamo le stringhe dei genitori nei punti di crossover, generando la nuova prole. Per esempio, il primo figlio della prima coppia riceve le prime tre cifre (327) dal primo genitore e quelle rimanenti (48552) dal secondo. Gli stati del problema a 8 regine coinvolti in questo passo di ricombinazione sono mostrati nella Figura 4.7.

Infine, in (e), ogni posizione nelle stringhe è soggetta a una mutazione casuale con una piccola probabilità indipendente. Nel nostro esempio, una cifra è mutata nella prima, terza e quarta delle stringhe della prole. Nel problema delle 8 regine, questo corrisponde a scegliere a caso una regina e spostarla altrettanto casualmente in una casella della stessa colonna. Spesso la popolazione è diversificata già all'inizio del processo, perciò il crossover spesso effettua grandi passi nello spazio degli stati all'inizio del processo (come nel simulated annealing). Dopo molte generazioni di selezione verso un valore della fitness più alto, la popolazione si fa meno diversificata e generalmente si effettuano passi più piccoli. La Figura 4.8 mostra un algoritmo che implementa tutti i passi descritti.

Gli algoritmi genetici sono simili alla ricerca beam stocastica, ma con l'aggiunta dell'operazione di crossover. Questo è un vantaggio se ci sono blocchi che eseguono funzioni utili.

```

function ALGORITMO-GENETICO(popolazione, fitness) returns un individuo
repeat
    pesi  $\leftarrow$  PESATO-DA(popolazione, fitness)
    popolazione2  $\leftarrow$  lista vuota
    for i = 1 to DIMENSIONE(popolazione) do
        genitore1, genitore 2  $\leftarrow$  SELEZIONE-CASUALE-PESATA(popolazione, pesi, 2)
        figlio  $\leftarrow$  RIPRODUZIONE(genitore1, genitore2)
        if (piccola probabilità casuale) then figlio  $\leftarrow$  MUTAZIONE(figlio)
        aggiungi figlio a popolazione2
    popolazione  $\leftarrow$  popolazione2
    until qualche individuo ha un valore della fitness sufficientemente alto, o è passato abbastanza tempo
    return l'individuo migliore nella popolazione in base alla fitness

function RIPRODUZIONE(genitore1, genitore2) returns un individuo
    n  $\leftarrow$  LUNGHEZZA(genitore1)
    c  $\leftarrow$  numero casuale fra 1 e n
    return CONCATENA(SOTTOSTRINGA(genitore1, 1, c), SOTTOSTRINGA(genitore2, c + 1, n))

```

Figura 4.8 Un algoritmo genetico. All'interno della funzione, *popolazione* è una lista ordinata di individui, *pesi* è una lista dei corrispondenti valori della fitness per ogni individuo e *fitness* è una funzione per calcolare quei valori.

Per esempio, può darsi che disporre le prime tre regine nelle posizioni 2, 4 e 6 (nelle quali non si attaccano tra loro) costituisca un blocco utile, che può essere combinato utilmente con altri blocchi utili che appaiono in altri individui per costruire una soluzione. Si può dimostrare matematicamente che, se i blocchi non hanno uno scopo preciso – per esempio, se le posizioni del codice genetico sono permutate a caso – allora il crossover non porta alcun vantaggio.

La teoria degli algoritmi genetici spiega questo meccanismo usando il concetto di **schemi**, una sottostringa in cui alcune posizioni possono essere lasciate non specificate. Per esempio, lo schema 246***** descrive tutti gli stati del problema delle 8 regine in cui le prime tre regine sono rispettivamente nelle posizioni 2, 4 e 6. Le stringhe che corrispondono allo schema (come 24613578) sono denominate **istanze** di quello schema. Si può dimostrare che, se il valore medio della fitness delle istanze di uno schema è superiore alla media, allora il numero di istanze di quello schema aumenterà nel tempo.

schemi

istanza

Chiaramente questo effetto non sarà significativo se i bit adiacenti sono totalmente scorrelati tra loro, perché in tal caso ci saranno pochi blocchi contigui capaci di fornire benefici consistenti. Gli algoritmi genetici funzionano meglio quando gli schemi corrispondono a componenti significative di una soluzione. Per esempio, se la stringa rappresenta un'antenna, gli schemi potrebbero rappresentare componenti quali riflettori e deflettori. Un buon componente probabilmente sarà utile in una varietà di progetti diversi. Questo suggerisce che il successo degli algoritmi genetici richiede una cura particolare nella progettazione della rappresentazione.

Nella pratica, gli algoritmi genetici hanno un ruolo significativo nell'ampio panorama dei metodi di ottimizzazione (Marler e Arora, 2004), in particolare per problemi strutturati complessi come la configurazione di circuiti o il job-shop scheduling, e più recentemente per l'evoluzione dell'architettura delle reti neurali deep (Miikkulainen *et al.*, 2019). Non è chiaro in quale grado l'interesse per gli algoritmi genetici nasca dalla loro superiorità per compiti specifici e in quale grado dal fatto che la metafora dell'evoluzione è molto attraente.

EVOLUZIONE E RICERCA

La teoria dell'**evoluzione** è stata sviluppata nel fondamentale libro di Charles Darwin *Sull'origine delle specie per mezzo della selezione naturale* (1859) e in modo indipendente da Alfred Russel Wallace (1858). L'idea centrale è semplice: nella riproduzione si verificano variazioni che saranno conservate nelle generazioni successive in modo proporzionale al loro effetto sul valore adattativo (fitness).

La teoria di Darwin fu sviluppata senza avere conoscenza dei meccanismi con cui i caratteri degli organismi possono essere ereditati o modificati. Le leggi probabilistiche che governano questi processi furono identificate per la prima volta da Gregor Mendel (1866), un monaco che fece esperimenti con i piselli dolci. Molto più tardi, Watson e Crick (1953) identificarono la struttura della molecola del DNA e il suo alfabeto, AGTC (adenina, guanina, timina e citosina). Nel modello standard, le variazioni si verificano sia per mutazioni puntiformi nella sequenza di lettere sia per “crossover” (per mezzo del quale il DNA di un individuo è generato combinando tra loro lunghe porzioni del DNA dei genitori).

Abbiamo già descritto le analogie tra questo modello e gli algoritmi di ricerca locale; la differenza principale tra la ricerca beam stocastica e l'evoluzione sta nel fatto che la riproduzione è sessuata e i successori sono generati non da uno, ma da *più* individui. I meccanismi reali dell'evoluzione, comunque, sono molto più ricchi di quelli degli algoritmi genetici. Per esempio, le mutazioni possono coinvolgere inversioni, duplicazioni e spostamenti di grandi blocchi di DNA; alcuni virus “prendono in prestito” DNA da un organismo e lo inseriscono in un altro; e ci sono geni trasponibili che non fanno altro che copiare se stessi migliaia di volte all'interno del genoma.

Esistono persino geni che avvelenano le cellule dei partner potenziali che non contengono una loro copia, aumentando così la propria probabilità di replicazione. La cosa più importante è che *i geni stessi codificano i meccanismi* usati per la riproduzione del genoma e la sua traduzione in un organismo; negli algoritmi genetici invece tali meccanismi sono racchiusi in un programma separato e non sono codificati nelle stringhe manipolate.

L'evoluzione di Darwin potrebbe sembrare inefficiente, avendo generato circa 10^{43} organismi alla cieca senza migliorare affatto le sue euristiche di ricerca. Mentre il grande naturalista francese Jean Lamarck (1809) si sbagliava nel sostenere che i caratteri acquisiti per adattamento durante la vita di un organismo sarebbero passati alla sua progenie, James Baldwin (1896) con la sua teoria a prima vista simile non si sbagliava: l'apprendimento può effettivamente rilassare il panorama della fitness, portando a un'accelerazione dell'evoluzione. Un organismo che ha un carattere non abbastanza adattativo per il suo ambiente, lo passa alla progenie se ha plasticità sufficiente per imparare ad adattarsi all'ambiente in un modo vantaggioso. Le simulazioni al computer (Hinton e Nowlan, 1987) confermano che questo **effetto Baldwin** è reale e che una conseguenza di ciò è che le cose più difficili da imparare vanno a finire nel genoma, mentre quelle più facili non devono necessariamente risiedervi (Morgan e Griffiths, 2015).

4.2 Ricerca locale in spazi continui

Nel Capitolo 2 abbiamo spiegato la differenza tra ambienti discreti e continui, specificando che la maggior parte degli ambienti reali appartengono alla seconda categoria. Uno spazio delle azioni continuo ha un fattore di ramificazione infinito, quindi non è affrontabile dalla maggior parte degli algoritmi descritti finora (fatta eccezione per l'hill climbing con prima scelta e il simulated annealing).

Questo paragrafo fornisce un'introduzione *molto breve* ad alcune tecniche di ricerca locale per spazi continui. La letteratura sull'argomento è vasta; alcuni dei metodi base hanno avuto origine nel XVII secolo, dopo lo sviluppo del calcolo differenziale da parte di Newton

e Leibniz.² Avremo l'occasione di usare queste tecniche in questo libro, nei capitoli sull'apprendimento, la visione e la robotica.

Cominciamo con un esempio. Supponiamo di voler costruire tre nuovi aeroporti in Romania, in posizioni tali da minimizzare la somma dei quadrati delle distanze in linea d'aria da ogni città sulla mappa all'aeroporto più vicino (cfr. Figura 3.1 con la mappa della Romania). Lo spazio degli stati sarà quindi definito dalle coordinate degli aeroporti: $(x_1, y_1), (x_2, y_2)$ e (x_3, y_3) . Questo spazio ha sei dimensioni; possiamo anche dire che gli stati sono definiti da sei **variabili** (in generale gli stati sono definiti da un vettore n -dimensionale \mathbf{x}). Muoversi in questo spazio significa spostare uno o più aeroporti sulla mappa. La funzione obiettivo $f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3)$ è abbastanza facile da calcolare in un particolare stato, una volta determinate le città più vicine. Sia C_i l'insieme delle città il cui aeroporto più vicino (nello stato \mathbf{x}) è i . Allora abbiamo:

$$f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2. \quad (4.1)$$

Questa espressione è corretta non solo per lo stato \mathbf{x} , ma anche per stati nell'intorno locale di \mathbf{x} . Non è però corretta globalmente: se ci allontaniamo troppo da \mathbf{x} (modificando di molto la posizione di uno o più degli aeroporti), allora l'insieme delle città più vicine a quell'aeroporto cambia, e dobbiamo ricalcolare C_i .

Un modo per affrontare uno spazio degli stati continuo consiste nel **discretizzarlo**. Per esempio, anziché lasciare che le posizioni (x_1, y_1) possano essere qualsiasi punto in uno spazio bidimensionale continuo, potremmo limitarle a punti fissi su una griglia rettangolare con spaziatura di lunghezza δ (delta). Allora, anziché avere un numero infinito di successori, ogni stato ne avrebbe soltanto 12, corrispondenti all'incremento di una delle sei variabili di un valore $\pm\delta$. Ora possiamo applicare tutti i nostri algoritmi di ricerca locale a questo spazio discreto. In alternativa, potremmo rendere finito il fattore di ramificazione mediante un campionamento casuale degli stati successori, muovendoci in una direzione casuale di una distanza piccola δ . I metodi che misurano il progresso compiuto in base alla variazione del valore della funzione obiettivo tra due punti vicini si chiamano metodi del **gradiente empirico**. La ricerca con gradiente empirico è simile all'hill climbing con ascesa più ripida in una versione discretizzata dello spazio degli stati. Riducendo il valore di δ nel tempo possiamo ottenere una soluzione più accurata, ma non c'è necessariamente convergenza a un ottimo globale nel passaggio al limite.

Spessoabbiamo una funzione obiettivo espressa in una forma matematica che ci consente di usare l'analisi matematica per risolvere il problema in modo analitico anziché empirico. Molti metodi cercano di usare il **gradiente** del panorama per trovare un massimo. Il gradiente della funzione obiettivo è un vettore ∇f che fornisce l'entità (modulo) e la direzione della pendenza più ripida. Per il nostro problema, abbiamo:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right).$$

In alcuni casi, possiamo trovare un massimo risolvendo l'equazione $\nabla f = 0$. Questo potrebbe essere fatto, per esempio, se stessimo piazzando un solo aeroporto; la soluzione sarebbe la media aritmetica delle coordinate di tutte le città. In molti casi, tuttavia, quest'equazione non può essere risolta in forma chiusa. Con tre aeroporti, per esempio, l'espressione del gra-

variabile

discretizzare

gradiente empirico

gradiente

² La conoscenza di vettori, matrici e derivate è utile per questo paragrafo (cfr. Appendice A).

diente dipende dalle città che sono più vicine a ogni aeroporto nello stato corrente. Questo significa che possiamo calcolare il gradiente *localmente* (ma non *globalmente*); per esempio,

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c). \quad (4.2)$$

Data un'espressione localmente corretta per il gradiente, possiamo ancora eseguire l'hill climbing seguendo la massima pendenza e aggiornando lo stato secondo la formula:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}),$$

dimensione del passo

line search

Newton–Raphson

dove α è una costante piccola, spesso chiamata **dimensione del passo**. Esistono numerosissimi metodi per determinare il valore di α . Il problema principale è che se α è troppo piccola sono richiesti troppi passi; se α è troppo grande, la ricerca potrebbe “arrivare lunga” e superare il punto di massimo. La tecnica della **line search** (ricerca unidimensionale) cerca di superare questo dilemma estendendo la direzione del gradiente (di solito raddoppiando progressivamente il valore di α) finché f non comincia a diminuire. Il punto in cui questo accade diventa il nuovo stato corrente. Fatto ciò, ci sono diverse scuole di pensiero circa il metodo da seguire per scegliere la nuova direzione.

Per molti problemi, l'algoritmo più efficace rimane l'antico metodo **Newton–Raphson**. Si tratta di una tecnica generale per trovare le radici delle funzioni, ovvero per risolvere equazioni nella forma $g(x) = 0$. Il metodo funziona calcolando una nuova stima della radice x secondo la formula di Newton:

$$x \leftarrow x - g(x)/g'(x).$$

Per trovare un massimo o un minimo di f , dobbiamo trovare una \mathbf{x} tale che il gradiente sia zero ($\nabla f(\mathbf{x}) = 0$). Quindi $g(x)$ nella formula di Newton diventa $\nabla f(\mathbf{x})$, e la funzione di aggiornamento può essere scritta in forma vettoriale come:

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x}).$$

Hessiana

dove $\mathbf{H}_f(\mathbf{x})$ è la matrice **Hessiana** delle derivate seconde, i cui elementi H_{ij} sono dati da $\partial^2 f / \partial x_i \partial x_j$. Per il nostro esempio dell'aeroporto, dall'Equazione (4.2) possiamo vedere che $\mathbf{H}_f(\mathbf{x})$ è particolarmente semplice: gli elementi al di fuori della diagonale sono zero e gli elementi della diagonale per l'aeroporto i sono pari al doppio del numero di città in C_i . Un rapido calcolo mostra che un passo dell'aggiornamento sposta l'aeroporto i direttamente sul centroide di C_i , il minimo dell'espressione locale per f dall'Equazione (4.1).³ Per problemi ad alto numero di dimensioni, tuttavia, calcolare gli n^2 componenti dell'Hessiana e invertirla potrebbe risultare costoso, perciò sono state sviluppate molte versioni approssimate del metodo Newton–Raphson.

I metodi di ricerca locale soffrono i massimi locali, le creste e i plateau negli spazi continui esattamente come in quelli discreti. Il riavvio casuale e il simulated annealing si rivelano spesso molto utili. Gli spazi continui a molte dimensioni, comunque, sono posti molto grandi dove è sempre facile perdersi.

ottimizzazione vincolata

Un ultimo argomento che è utile conoscere è l'**ottimizzazione vincolata**. Un problema di ottimizzazione è vincolato se le soluzioni devono soddisfare dei vincoli esplicativi sui valori di ogni variabile. Per esempio, nel nostro problema di posizionamento di aeroporti, potremmo formulare i vincoli che i siti siano all'interno della Romania e sulla terraferma (anziché nel

³ In generale, l'aggiornamento di Newton-Raphson può essere visto come l'adattamento di una superficie quadratica a f in \mathbf{x} seguito dallo spostamento diretto sul minimo di tale superficie, che è anche il minimo di f , se f è quadratica.

mezzo di un lago). La difficoltà dei problemi di ottimizzazione vincolata dipende dalla natura dei vincoli e della funzione obiettivo. La categoria meglio conosciuta è quella dei problemi di **programmazione lineare**, in cui i vincoli sono espressi da diseguaglianze lineari che formano un **insieme convesso**⁴ e la funzione obiettivo è anch'essa lineare. La complessità temporale della programmazione lineare è polinomiale nel numero di variabili.

La programmazione lineare è probabilmente il metodo di ottimizzazione più studiato e di più ampia utilità. È un caso speciale del più generale problema di **ottimizzazione convessa**, in cui l'area vincolata può essere qualsiasi area convessa e l'obiettivo qualsiasi funzione convessa nell'area vincolata. In certe condizioni, i problemi di ottimizzazione convessa sono anche risolvibili polinomialmente e la risoluzione potrebbe essere fattibile nella pratica con migliaia di variabili. Diversi importanti problemi nei campi dell'apprendimento automatico e della teoria del controllo possono essere formulati come problemi di ottimizzazione convessa.

**programmazione
lineare**
insieme convesso

**ottimizzazione
convessa**

4.3 Ricerca con azioni non deterministiche

Nel Capitolo 3 abbiamo ipotizzato che l'ambiente fosse completamente osservabile, deterministico e noto, quindi che un agente potesse osservare lo stato iniziale, calcolare una sequenza di azioni che raggiungesse l'obiettivo ed eseguirle “a occhi chiusi” senza usare le sue percezioni.

Quando l'ambiente è parzialmente osservabile, invece, l'agente non sa con sicurezza in quale stato si trova; e quando l'ambiente è non deterministico, l'agente non sa in quale stato arriverà dopo l'esecuzione di un'azione. Questo significa che anziché pensare “Sono nello stato s_1 e se eseguo l'azione a andrò nello stato s_2 ” l'agente penserà “Sono nello stato s_1 o s_3 , e se eseguo l'azione a passerò nello stato s_2, s_4 o s_5 ”. Chiamiamo **stato-credenza** un insieme di stati fisici che l'agente ritiene siano possibili.

stato-credenza

In ambienti parzialmente osservabili e non deterministici, la soluzione di un problema non è una sequenza ma un **piano condizionale** (piano di contingenza o **strategia**) che specifica che cosa fare in base alle percezioni ricevute dall'agente durante l'esecuzione del piano. In questo paragrafo esaminiamo il caso del non determinismo, nel Paragrafo 4.4 tratteremo il caso dell'osservabilità parziale.

piano condizionale

4.3.1 Il mondo dell'aspirapolvere erratico

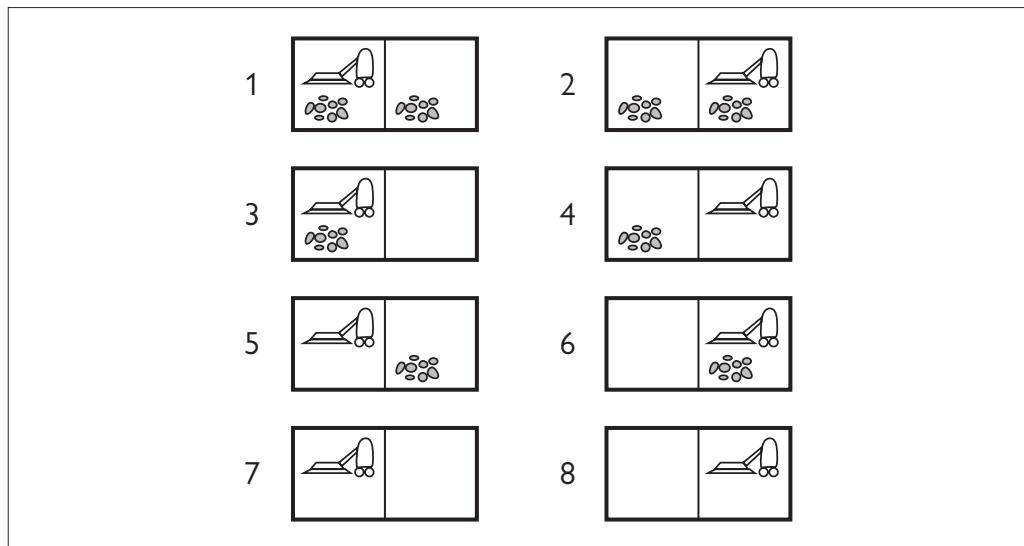
Il mondo dell'aspirapolvere, presentato nel Capitolo 2, ha otto stati, come mostrato nella Figura 4.9. Ci sono tre azioni, *Sinistra*, *Destra* e *Aspira*, e l'obiettivo è ripulire tutto lo sporco (stati 7 e 8). Se l'ambiente è completamente osservabile, deterministico e interamente noto, il problema si risolve facilmente con uno qualsiasi degli algoritmi descritti nel Capitolo 3 e la soluzione è una sequenza di azioni. Per esempio, se lo stato iniziale è 1, la sequenza di azioni [*Aspira*,*Destra*,*Aspira*] raggiungerà uno stato obiettivo, 8.

Ora supponiamo di introdurre il non determinismo nella forma di un potente ma erratico aspirapolvere. Nel **mondo dell'aspirapolvere erratico**, l'azione *Aspira* opera come segue:

- quando è applicata a un riquadro sporco, l'azione lo pulisce e talvolta pulisce anche un riquadro adiacente;

⁴ Un insieme di punti \mathcal{S} è convesso se la linea che unisce due punti qualsiasi di \mathcal{S} è anch'essa contenuta in \mathcal{S} . Una **funzione convessa** è una funzione per cui lo spazio “sopra” di essa forma un insieme convesso; per definizione, le funzioni convesse non hanno minimi locali (al contrario dei minimi globali).

Figura 4.9 Gli otto stati possibili del mondo dell'aspirapolvere; gli stati 7 e 8 sono stati obiettivo.



- quando è applicata a un riquadro pulito, l'azione talvolta deposita dello sporco sul tappeto.⁵

Per fornire una formulazione precisa di questo problema, dobbiamo generalizzare la nozione di **modello di transizione** presentata nel Capitolo 3. Invece di definire il modello di transizione con una funzione **RISULTATO** che restituisce un singolo stato, utilizziamo una funzione **RISULTATI** che restituisce un insieme di possibili stati risultato. Per esempio, nel mondo dell'aspirapolvere erratico, l'azione *Aspira* nello stato 1 pulisce soltanto la posizione corrente o entrambe le posizioni:

$$\text{RISULTATI}(1, \text{Aspira}) = \{5, 7\}$$

Se iniziamo nello stato 1, nessuna singola *sequenza* di azioni risolve il problema, mentre il seguente **piano condizionale** lo risolve:

$$[\text{Aspira}, \text{if } \text{Stato} = 5 \text{ then } [\text{Destra}, \text{Aspira}] \text{ else } []] . \quad (4.3)$$

Vediamo qui che un piano condizionale può contenere passi **if-then-else**; ciò significa che le soluzioni sono *alberi* e non sequenze. In questo caso la condizione nell'istruzione **if** verifica qual è lo stato corrente; si tratta di un'informazione che l'agente sarà in grado di osservare al momento dell'esecuzione, ma non conosce al momento della pianificazione. In alternativa, avremmo potuto utilizzare una formulazione che testasse la percezione anziché lo stato. Molti problemi del mondo fisico reale sono problemi di contingenza, perché una previsione esatta del futuro è impossibile. Per questo motivo, molte persone tengono gli occhi aperti quando camminano.

4.3.2 Alberi di ricerca AND-OR

Ora ci chiediamo come trovare soluzioni contingenti a problemi non deterministici. Come nel Capitolo 3, iniziamo costruendo alberi di ricerca, ma in questo caso gli alberi hanno un carattere diverso. In un ambiente deterministico, l'unica ramificazione è introdotta dalle

⁵ Supponiamo che la maggior parte dei lettori affronti problemi simili e possa simpatizzare con il nostro agente. Ci scusiamo con coloro che possiedono apparecchi moderni ed efficienti che non traranno vantaggio da questo strumento pedagogico.

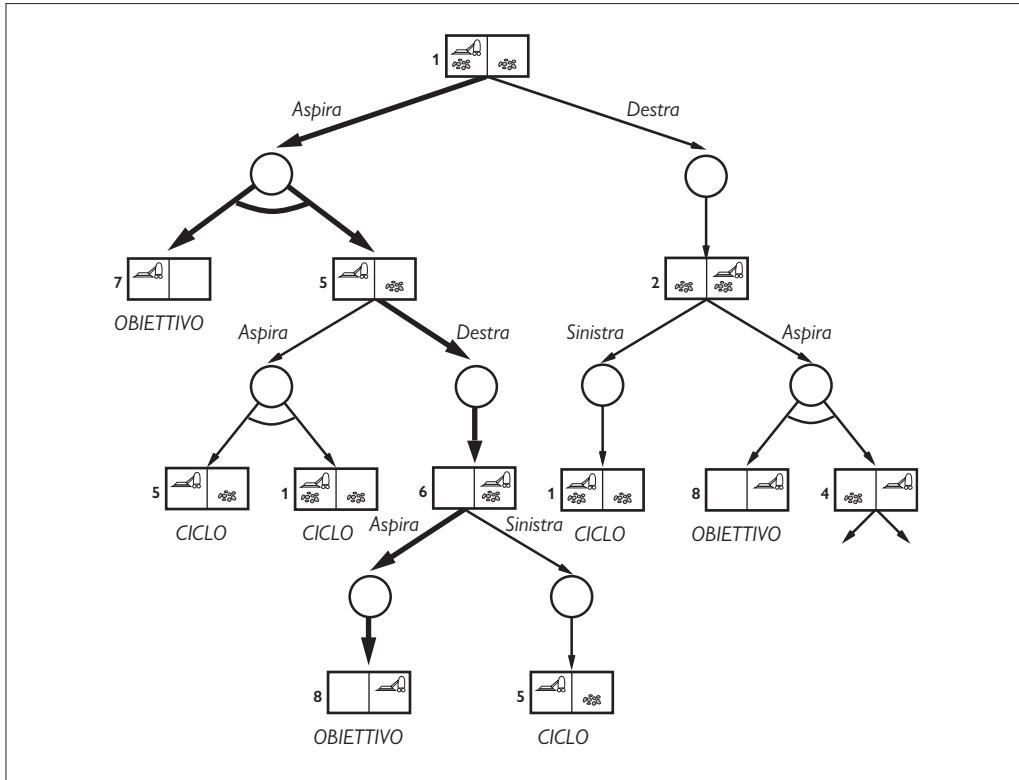


Figura 4.10 I primi due livelli dell'albero di ricerca per il mondo dell'aspirapolvere erratico. I nodi degli stati sono nodi OR in cui deve essere scelta un'azione. Nei nodi AND, mostrati come cerchi, ogni risultato deve essere gestito, come indicato dall'arco che collega i rami in uscita. La soluzione trovata è mostrata dalle linee più spesse.

scelte dell'agente in ogni stato: “Posso fare questa azione o quella”; parliamo in questo caso di **nodi OR**. Nel mondo dell'aspirapolvere, per esempio, in un nodo OR l'agente sceglie *Sinistra* o *Destra* o *Aspira*. In un ambiente non deterministico, la ramificazione è anche legata alla scelta del risultato per ogni azione, effettuata dall'*ambiente*. In questo caso parliamo di **nodi AND**. Per esempio, l'azione *Aspira* nello stato 1 porta a uno stato-credenza {5, 7}, perciò l'agente dovrebbe trovare un piano per lo stato 5 e per lo stato 7. Questi due tipi di nodi si alternano, generando un **albero AND-OR** come quello illustrato nella Figura 4.10.

Una soluzione per un problema di ricerca AND-OR è un sottoalbero dell'albero di ricerca completo che (1) ha un nodo obiettivo in ogni foglia, (2) specifica una sola azione in ognuno dei suoi nodi OR e (3) include ogni ramo uscente da ognuno dei suoi nodi AND. La soluzione è mostrata dalle linee più spesse nella figura, e corrisponde al piano fornito nell'Equazione (4.3)

La Figura 4.11 illustra un algoritmo ricorsivo in profondità per la ricerca su un grafo AND-OR. Un aspetto chiave dell'algoritmo è il modo in cui gestisce i cicli, che spesso si presentano in problemi non deterministici (per esempio, se un'azione talvolta non ha effetto, o se un effetto indesiderato può essere corretto). Se lo stato corrente è identico a uno stato sul cammino dalla radice, allora l'algoritmo termina con un fallimento. Questo non significa che non esiste *alcuna* soluzione dallo stato corrente, ma soltanto che, se *esiste* una soluzione non ciclica, deve essere raggiungibile dalla precedente incarnazione dello stato corrente, perciò la nuova incarnazione può essere scartata. Con questo controllo ci assicuriamo che l'algoritmo termini in ogni spazio degli stati finito, perché ogni cammino deve raggiungere un

nodo OR

nodo AND

albero AND-OR

```

function RICERCA-AND-OR(problema) returns un piano condizionale, o fallimento
  return RICERCA-OR(problema, problema.STATOINIZIALE, [ ])

function RICERCA-OR(problema, stato, cammino) returns un piano condizionale, o fallimento
  if problema.È-OBIETTIVO(stato) then return il piano vuoto
  if È-CICLO(cammino) then return fallimento
  for each azione in problema.AZIONI(stato) do
    piano  $\leftarrow$  RICERCA-AND(problema, RISULTATI(stato, azione), [stato + cammino])
    if piano  $\neq$  fallimento then return [azione + piano]
  return fallimento

function RICERCA-AND(problema, stati, cammino) returns un piano condizionale, o fallimento
  for each si in stati do
    pianoi  $\leftarrow$  RICERCA-OR(problema, si, cammino)
    if pianoi = fallimento then return fallimento
  return [if s1 then piano1 else if s2 then piano2 else . . . if sn-1 then pianon-1 else pianon]

```

Figura 4.11 Un algoritmo per la ricerca su grafo and-or generato da ambienti non deterministici. Una soluzione è un piano condizionale che considera ogni risultato non deterministico e ha un piano per ognuno.

obiettivo, un vicolo cieco o uno stato ripetuto. Notate che l'algoritmo non controlla se lo stato corrente è una ripetizione di uno stato in qualche *altro* cammino dalla radice, cosa importante per l'efficienza.

I grafi AND-OR possono essere esplorati dai metodi in ampiezza o best-first. Il concetto di funzione euristica deve essere modificato per stimare il costo di una soluzione contingente anziché di una sequenza, ma la nozione di ammissibilità viene mantenuta ed esiste un analogo dell'algoritmo A* per trovare soluzioni ottime (cfr. le note bibliografiche al termine di questo capitolo).

4.3.3 Prova, prova ancora

soluzione ciclica

Consideriamo un mondo dell'aspirapolvere *scivoloso*, identico a quello dell'aspirapolvere normale (non erratico) fatta eccezione per il fatto che talvolta le azioni di movimento falliscono, lasciando l'agente nella medesima posizione. Per esempio, con l'azione *Destra* nello stato 1 si arriva allo stato-credenza {1,2}. La Figura 4.12 mostra parte del grafo di ricerca; è chiaro che non ci sono soluzioni acicliche dallo stato 1, e RICERCA-AND-OR terminerebbe fallendo. Esiste però una **soluzione ciclica**, quella di continuare a provare *Destra* finché funziona. Possiamo esprimere questa soluzione con un nuovo costrutto **while**:

[*Aspira, while Stato = 5 do Destra, Aspira*]

oppure aggiungendo un'**etichetta** (*label*) per denotare una specifica porzione del piano facendovi riferimento in seguito:

[*Aspira, L₁ : Destra, if Stato = 5 then L₁ else Aspira*] .

Quando un piano ciclico è una soluzione? Una condizione minima è che ogni foglia sia uno stato obiettivo e che una foglia sia raggiungibile da qualsiasi punto nel piano. Oltre a questo, dobbiamo considerare il non determinismo. Se il meccanismo di guida dell'aspirapolvere robotizzato funziona per un po' di tempo, ma in altri momenti slitta in modo casuale e indi-

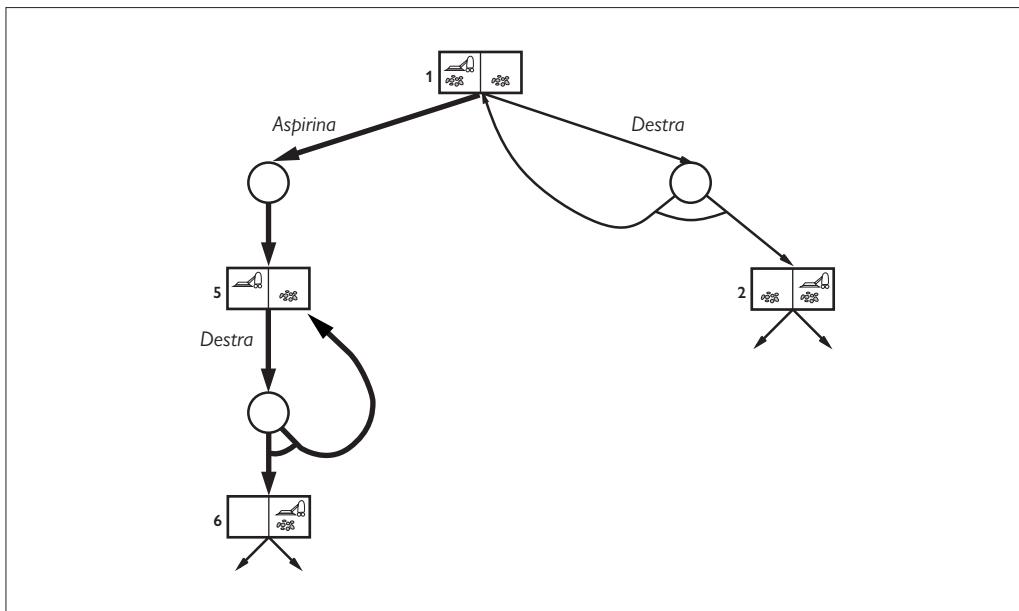


Figura 4.12
Parte del grafo di ricerca per il mondo dell'aspirapolvere scivoloso, in cui abbiamo evidenziato (alcuni) cicli. Tutte le soluzioni per questo problema sono piani ciclici perché non vi è modo per spostarsi in maniera affidabile.

pendente, allora l'agente può essere fiducioso che, se l'azione viene ripetuta per un numero di volte sufficiente, alla fine funzionerà e il piano avrà successo. Ma se il non determinismo è dovuto a un fatto non osservato circa il robot o l'ambiente, per esempio una cinghia di trasmissione che si rompe e impedisce al robot di muoversi, allora la ripetizione dell'azione non è di alcun aiuto.

Un modo per capire questa decisione è dire che la formulazione iniziale del problema (completamente osservabile, non deterministico) è abbandonata in favore di una formulazione diversa (parzialmente osservabile, deterministico) dove il fallimento del piano ciclico è attribuito a una proprietà non osservata della cinghia di trasmissione. Nel Capitolo 12 vedremo come decidere quali delle diverse possibilità incerte è la più probabile.

4.4 Ricerca con osservazioni parziali

Passiamo ora al problema dell'osservabilità parziale, in cui le percezioni dell'agente non sono sufficienti per determinare lo stato esatto. Questo significa che alcune delle azioni dell'agente punteranno a ridurre l'incertezza riguardo lo stato corrente.

4.4.1 Ricerca in assenza di osservazioni

Quando le osservazioni dell'agente *non forniscono alcuna informazione*, abbiamo un **problema senza sensori**, talvolta detto problema **conforme**. A prima vista si potrebbe pensare che l'agente privo di sensori non abbia speranza di risolvere un problema se non ha idea dello stato da cui inizia, ma le soluzioni senza sensori sono sorprendentemente comuni e utili, principalmente perché *non* si affidano al buon funzionamento dei sensori. Nei sistemi di produzione industriale, per esempio, sono stati sviluppati molti metodi ingegnosi per orientare correttamente dei pezzi partendo da una posizione iniziale sconosciuta, utilizzando una sequenza di azioni senza impiegare sensori. A volte un piano senza sensori è la scelta migliore anche quando è disponibile un piano condizionale con sensori. Per esempio, i medici spesso prescrivono antibiotici a largo spettro anziché utilizzare il piano condizionale di

problema senza
sensori
conforme

forzare

effettuare un esame del sangue, attendere i risultati e poi prescrivere un antibiotico più specifico. Il piano senza sensori consente di risparmiare tempo e denaro ed evita il rischio che l'infezione peggiori prima che siano disponibili i risultati dell'esame.

Consideriamo ora una versione senza sensori del mondo dell'aspirapolvere (deterministico). Supponiamo che l'agente conosca la geografia del suo mondo, ma non la propria posizione o la distribuzione della sporcizia. In tal caso, il suo stato iniziale potrebbe essere qualsiasi elemento dell'insieme {1,2,3,4,5,6,7,8} (Figura 4.9). Ora, se l'agente si muove a *Destra* arriverà in uno degli stati {2,4,6,8}, guadagnando informazioni senza alcuna percezione. Dopo [*Destra,Aspira*] l'agente terminerà sempre in uno degli stati {4,8}. Infine, dopo [*Destra,Aspira,Sinistra,Aspira*] l'agente raggiungerà sempre lo stato obiettivo 7, indipendentemente dallo stato di partenza. Diciamo che l'agente può **forzare** il mondo nello stato 7.

La soluzione di un problema senza sensori è una sequenza di azioni, non un piano condizionale (perché non vi è percezione). Ma la ricerca viene effettuata nello spazio degli stati-credenza e non degli stati fisici.⁶ Nello spazio degli stati-credenza il problema è *completamente osservabile* perché l'agente conosce sempre il proprio stato-credenza. Inoltre, la soluzione (se esiste) di un problema senza sensori è sempre una sequenza di azioni perché, come nei problemi tradizionali del Capitolo 3, le percezioni ricevute dopo ciascuna azione sono del tutto prevedibili: sono sempre vuote! Perciò non ci sono contingenze di cui tenere conto. Questo è vero *anche se l'ambiente è non deterministico*.

Potremmo introdurre nuovi algoritmi per problemi di ricerca senza sensori, ma invece di fare ciò, possiamo usare gli algoritmi già visti nel Capitolo 3 se trasformiamo il problema fisico sottostante in un problema di stati-credenza, in cui effettuiamo la ricerca in stati-credenza anziché stati fisici. Il problema originale *P* ha come componenti $AZIONI_P$, $RISULTATO_P$ e così via, e il problema di stati-credenza ha i componenti seguenti.

- **Stati:** lo spazio degli stati-credenza contiene ogni possibile sottoinsieme degli stati fisici. Se *P* ha N stati, allora il problema di stati-credenza ha 2^N stati-credenza, anche se molti potrebbero essere irraggiungibili dallo stato iniziale.
- **Stato iniziale:** è tipicamente lo stato-credenza costituito da tutti gli stati in *P*, anche se in alcuni casi l'agente avrà una conoscenza maggiore di questa.
- **Azioni:** questo è un aspetto delicato. Supponiamo che l'agente si trovi nello stato-credenza $b = \{s_1, s_2\}$, ma $AZIONI_P(s_1) \neq AZIONI_P(s_2)$; allora l'agente non sa quali azioni sono legali. Se supponiamo che le azioni illegali non abbiano alcun effetto sull'ambiente, possiamo effettuare l'*unione* di tutte le azioni in uno qualsiasi degli stati fisici contenuti nello stato-credenza corrente b :

$$AZIONI(b) = \bigcup_{s \in b} AZIONI_P(s).$$

D'altra parte, se un'azione illegale potesse portare alla catastrofe, sarebbe più sicuro consentire soltanto l'*intersezione*, considerando l'insieme delle azioni legali in *tutti* gli stati. Per il mondo dell'aspirapolvere, ogni stato presenta le stesse azioni legali, perciò entrambi i metodi forniscono lo stesso risultato.

- **Modello di transizione:** per azioni deterministiche, il nuovo stato-credenza contiene un solo stato risultato per ognuno dei possibili stati correnti (anche se alcuni stati risultato potrebbero coincidere):

$$b' = RISULTATO(b, a) = \{s' : s' = RISULTATO_P(s, a) \text{ e } s \in b\}. \quad (4.4)$$

⁶ In un ambiente interamente osservabile, ogni stato-credenza contiene un solo stato fisico. Perciò possiamo considerare gli algoritmi di ricerca del Capitolo 3 come se cercassero in uno spazio degli stati-credenza composto da stati-credenza che sono singoletti.

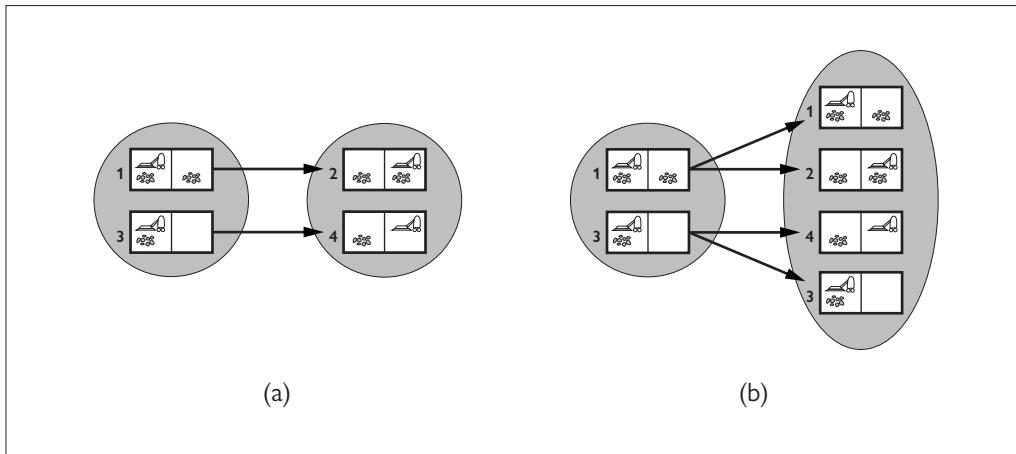


Figura 4.13 (a) Predizione del successivo stato-credenza per il mondo dell'aspirapolvere senza sensori con un'azione deterministica, Destra. (b) Predizione per lo stesso stato-credenza e la stessa azione nella versione scivolosa del mondo dell'aspirapolvere senza sensori.

Nel caso di non determinismo, il nuovo stato-credenza consiste di tutti i possibili risultati ottenuti applicando l'azione a uno qualsiasi degli stati nello stato-credenza corrente:

$$\begin{aligned} b' = \text{RISULTATO}(b, a) &= \{s' : s' \in \text{RISULTATI}_P(s, a) \text{ e } s \in b\} \\ &= \bigcup_{s \in b} \text{RISULTATI}_P(s, a), \end{aligned}$$

La dimensione di b' sarà uguale o minore di b per azioni deterministiche, ma potrebbe essere maggiore di b con azioni non deterministiche (Figura 4.13).

- **Test obiettivo:** l'agente può raggiungere l'obiettivo se *qualche* stato s nello stato-credenza soddisfa il test obiettivo del problema sottostante, $\text{È-OBIETTIVO}_P(s)$. L'agente raggiunge *sicuramente* l'obiettivo se *ogni* stato soddisfa $\text{È-OBIETTIVO}_P(s)$. Noi puntiamo a raggiungere sicuramente l'obiettivo.
- **Costo di azione:** anche questo è un punto delicato. Se la stessa azione può avere costi diversi in stati differenti, allora il costo di eseguire un'azione in un dato stato-credenza potrebbe essere uno tra diversi valori (nasce così una nuova classe di problemi, che esamineremo nell'Esercizio 4.MVAL). Per ora ipotizziamo che il costo di un'azione sia sempre lo stesso in tutti gli stati e così possa essere trasferito direttamente dal problema fisico sottostante.

La Figura 4.14 mostra lo spazio degli stati-credenza raggiungibili per il mondo dell'aspirapolvere senza sensore e deterministico. Ci sono soltanto 12 stati-credenza raggiungibili su un totale di $2^8 = 256$ stati-credenza possibili.

Le definizioni precedenti consentono di costruire automaticamente la formulazione del problema di ricerca nello spazio degli stati-credenza partendo dalla definizione del problema fisico sottostante. Una volta fatto ciò, possiamo risolvere problemi senza sensori utilizzando uno degli algoritmi di ricerca del Capitolo 3.

Nella ricerca su grafo normale, i nuovi stati raggiunti vengono controllati per determinare se sono già stati raggiunti in precedenza. Questo metodo funziona anche per gli stati-credenza; per esempio, nella Figura 4.14 la sequenza di azioni [Aspira, Sinistra, Aspira] iniziando dallo stato iniziale raggiunge lo stesso stato-credenza di [Destra, Sinistra, Aspira], ovvero {5,7}. Ora consideriamo lo stato-credenza raggiunto da [Sinistra], vale a dire {1,3,5,7}; ovviamente non è identico a {5,7}, ma è un *soprainsieme* di questo. Possiamo scartare (o potare) ogni stato soprainsieme come quello, poiché una soluzione da {1,3,5,7} deve essere una soluzione

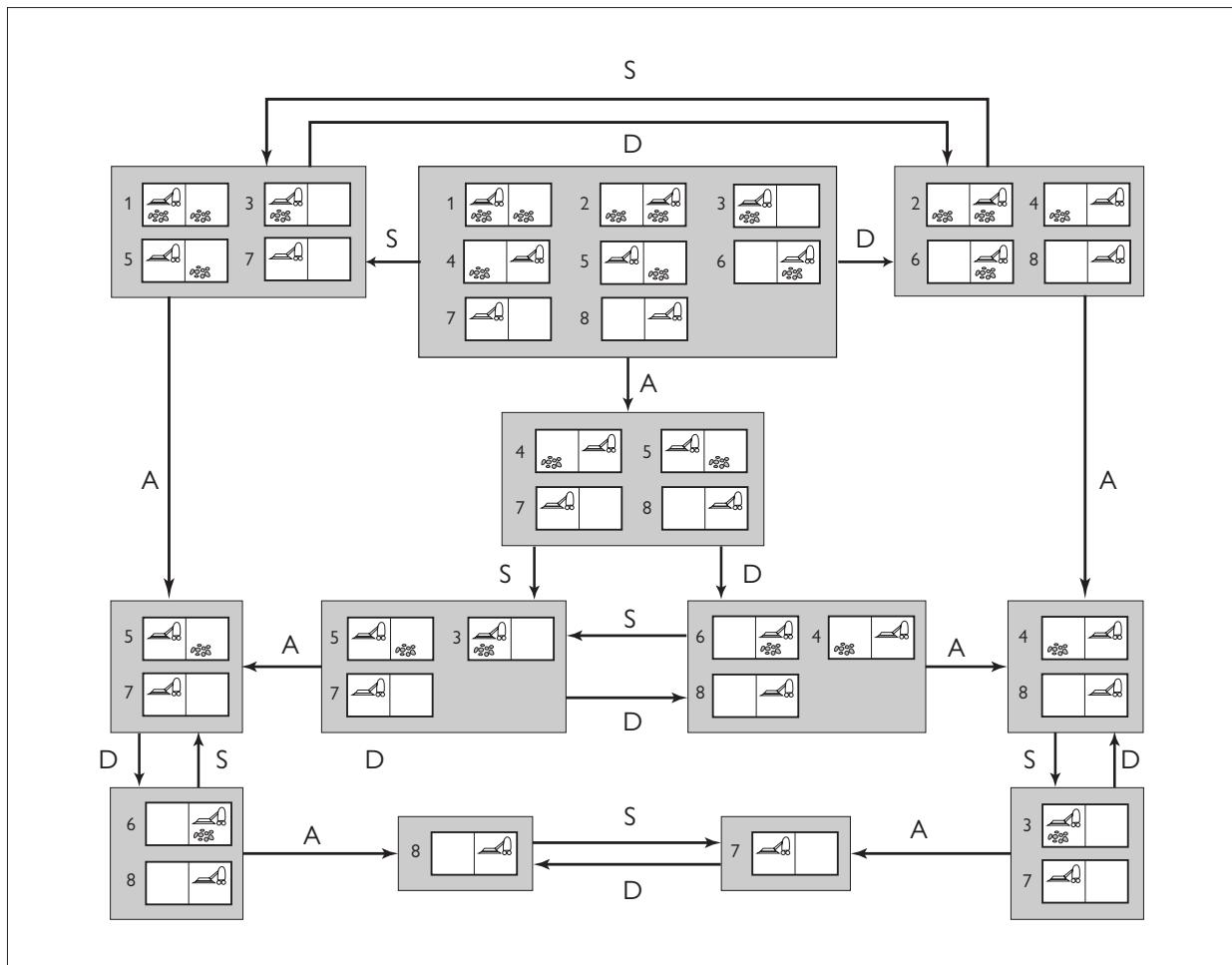


Figura 4.14 La porzione raggiungibile dello spazio degli stati-credenza per un mondo dell’aspirapolvere deterministico e senza sensori. Ogni riquadro rettangolare corrisponde a un singolo stato-credenza. In ogni momento l’agente ha uno stato-credenza ma non conosce lo stato fisico in cui si trova. Lo stato-credenza iniziale (totale ignoranza) corrisponde al rettangolo centrale in alto nella figura.

per ognuno dei singoli stati 1, 3, 5 e 7, quindi è una soluzione anche per qualsiasi combinazione di questi stati, come {5,7}; non ci serve dunque cercare di risolvere {1,3,5,7}, possiamo concentrarci sul risolvere lo stato-credenza strettamente più facile {5,7}.

Viceversa, se {1,3,5,7} è già stato generato ed è risultato risolvibile, allora qualsiasi suo *sottoinsieme*, come {5,7}, è certamente risolvibile (se ho una soluzione che funziona quando sono molto confuso circa lo stato in cui mi trovo, funzionerà anche quando sarò meno confuso). Questo ulteriore livello di potatura può migliorare notevolmente l’efficienza della risoluzione di problemi senza sensori.

Anche con questo miglioramento, tuttavia, la risoluzione di problemi senza sensori che abbiamo descritto finora è raramente applicabile nella pratica. Una difficoltà è la vastità dello spazio degli stati-credenza, abbiamo visto nel capitolo precedente che spesso uno spazio di dimensione N è troppo grande, e ora abbiamo spazi di ricerca di dimensione 2^N . Inoltre, ogni elemento dello spazio di ricerca è un insieme che può avere fino a N elementi; per N grande non saremo in grado di rappresentare nemmeno un singolo stato-credenza senza esaurire la memoria disponibile.

Una soluzione è quella di rappresentare lo stato-credenza mediante una descrizione più compatta. In italiano potremmo dire che l'agente non sa “Niente” nello stato iniziale; dopo uno spostamento a *Sinistra* potremmo dire: “Non nella colonna più a destra” e così via. Nel Capitolo 7 viene spiegato come fare ciò in uno schema di rappresentazione formale. Un altro approccio è quello di evitare gli algoritmi di ricerca standard, che trattano gli stati-credenza come “scatole nere” alla pari di qualsiasi altro stato, ed esaminare invece l'*interno* degli stati-credenza per sviluppare algoritmi di **ricerca stato-credenza incrementale** che costruiscano la soluzione considerando uno stato fisico per volta. Per esempio, nel mondo dell'aspirapolvere senza sensori, lo stato-credenza iniziale è $\{1,2,3,4,5,6,7,8\}$ e dobbiamo trovare una sequenza di azioni che funzioni in tutti e 8 gli stati. Possiamo farlo trovando una soluzione che funzioni per lo stato 1 e verificando se funziona per lo stato 2: se non funziona, torniamo indietro e troviamo una diversa soluzione per lo stato 1, e così via. Esattamente come una ricerca AND-OR deve trovare una soluzione per ogni ramo in un nodo AND, questo algoritmo deve trovare una soluzione per ogni stato nello stato-credenza. La differenza è che la ricerca AND-OR può trovare una soluzione diversa per ogni ramo, mentre una ricerca stato-credenza incrementale deve trovare *una sola* soluzione che funzioni per *tutti* gli stati.

**algoritmo di ricerca
stato-credenza
incrementale**

Il principale vantaggio dell'approccio incrementale è che generalmente è in grado di rilevare rapidamente il fallimento: quando uno stato-credenza è irrisolvibile, solitamente avviene che anche un piccolo sottoinsieme dello stato-credenza, costituito dai primi stati esaminati, è irrisolvibile. In alcuni casi questo porta a una velocizzazione proporzionale alla dimensione degli stati-credenza, che a loro volta potrebbero essere grandi quanto lo spazio degli stati fisici.

4.4.2 Ricerca in ambienti parzialmente osservabili

Molti problemi non si possono risolvere senza sensori. Per esempio, il rompicapo a 8 tasselli senza sensori è impossibile da risolvere. D'altra parte, anche pochi sensori possono fare molto: siamo in grado di risolvere rompicapi a 8 tasselli se riusciamo a vedere già solo il riquadro nell'angolo superiore sinistro. La soluzione richiede di spostare ciascun tassello a turno nel riquadro visibile, tenendo traccia della posizione da lì in poi.

Nel caso di un problema parzialmente osservabile, la specifica del problema includerà una funzione $\text{PERCEZIONE}(s)$ che restituisce la percezione ricevuta dall'agente in un dato stato. Se i sensori sono non deterministici, possiamo utilizzare una funzione PERCEZIONI che restituisce un insieme di possibili percezioni. Per problemi completamente osservabili, $\text{PERCEZIONE}(s) = s$ per ogni stato s , mentre per problemi senza sensori $\text{PERCEZIONE}(s) = \text{null}$.

Consideriamo un mondo dell'aspirapolvere con sensori locali, in cui l'agente è dotato di un sensore di posizione che restituisce la percezione S quando si trova nel riquadro sinistro, D quando si trova in quello destro e di un sensore di sporcizia che restituisce *Sporco* quando il riquadro corrente è sporco e *Pulito* quando è pulito. Allora la PERCEZIONE nello stato 1 è $[S, Sporco]$. In caso di osservabilità parziale, solitamente diversi stati producono la stessa percezione; lo stato 3 produrrà anch'esso $[S, Sporco]$. Quindi, data questa percezione iniziale, lo stato-credenza iniziale sarà $\{1,3\}$. Possiamo pensare che il modello di transizione tra stati-credenza per problemi parzialmente osservabili operi in tre fasi, come illustrato nella Figura 4.15.

- La fase di **predizione** calcola lo stato-credenza risultante dall'azione, $\text{RISULTATO}(b,a)$, esattamente come nei problemi senza sensori. Per sottolineare il fatto che si tratta di una predizione, utilizziamo la notazione $\hat{b} = \text{RISULTATO}(b,a)$, dove il “circonflesso” sopra b significa “stimato”, e usiamo anche $\text{PREDIZIONE}(b,a)$ come sinonimo di $\text{RISULTATO}(b,a)$.
- La fase delle **percezioni possibili** calcola l'insieme delle percezioni che potrebbero essere osservate nello stato-credenza predetto (utilizziamo la lettera o per osservazione):

$$\text{PERCEZIONI-POSSIBILI}(\hat{b}) = \{o : o = \text{PERCEZIONE}(s) \text{ e } s \in \hat{b}\}.$$

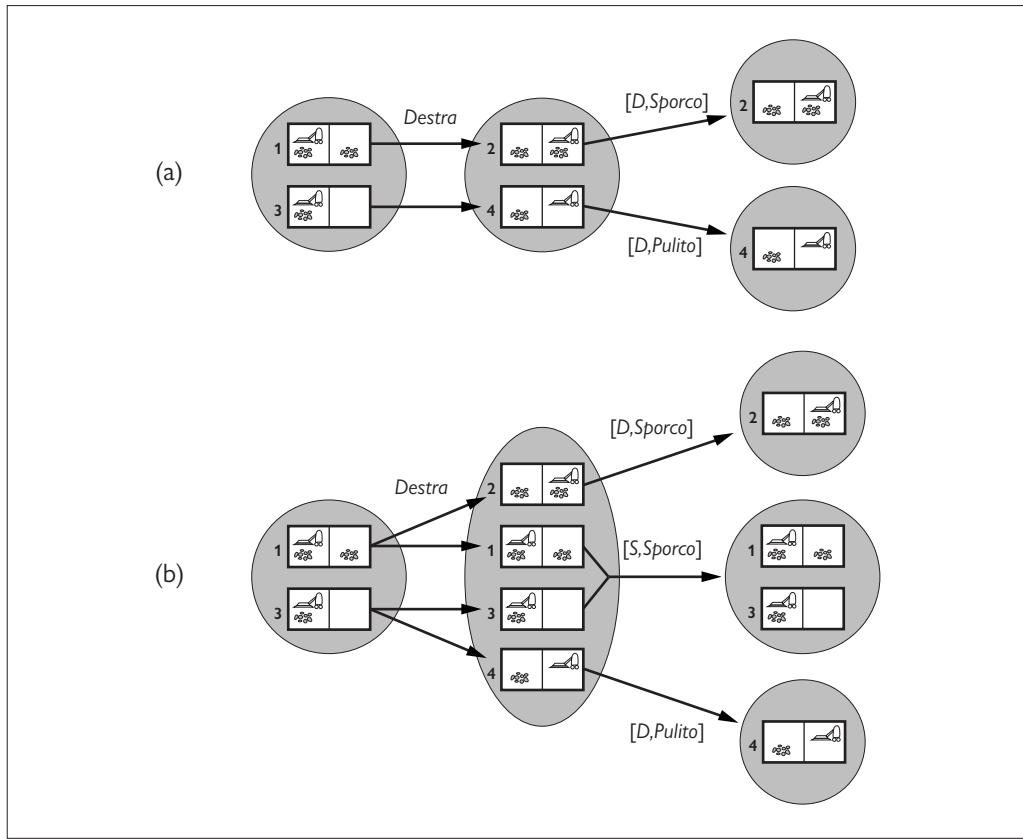


Figura 4.15 Due esempi di transizioni in mondi dell’aspirapolvere con sensori locali. (a) Nel mondo deterministico, *Destra* è applicata nello stato-credenza iniziale e porta a un nuovo stato-credenza predetto con due possibili stati fisici; per tali stati, le possibili percezioni sono $[D, \text{Sporco}]$ e $[D, \text{Pulito}]$, che portano a due stati-credenza, ognuno dei quali è un singoletto. (b) Nel mondo scivoloso, *Destra* è applicata nello stato-credenza iniziale e porta a un nuovo stato-credenza con quattro possibili stati fisici; per tali stati, le possibili percezioni sono $[S, \text{Sporco}]$, $[D, \text{Sporco}]$ e $[D, \text{Pulito}]$, che portano a tre stati-credenza, come mostrato.

- La fase di **aggiornamento** calcola, per ogni possibile percezione, lo stato-credenza che risulterebbe da essa. Lo stato-credenza aggiornato b_o è semplicemente l’insieme degli stati di \hat{b} che potrebbero aver prodotto la percezione:

$$b_o = \text{AGGIORNA}(\hat{b}, o) = \{s : o = \text{PERCEZIONE}(s) \text{ e } s \in \hat{b}\}.$$

L’agente deve gestire percezioni *possibili* durante la pianificazione, perché non può conoscere le percezioni *effettive* fino al momento dell’esecuzione del piano. Notate che il non determinismo nell’ambiente fisico può ingrandire lo stato-credenza nella fase di predizione, ma ogni stato-credenza aggiornato b_o non può essere più grande dello stato-credenza predetto \hat{b} ; le osservazioni possono soltanto aiutare a ridurre l’incertezza. Inoltre, per sensori deterministici, gli stati-credenza per le diverse percezioni possibili saranno disgiunti, formando una *partizione* dello stato-credenza predetto originario.

Mettendo insieme tutte e tre le fasi otteniamo i possibili stati-credenza risultanti da una data azione e le conseguenti percezioni possibili:

$$\begin{aligned} \text{RISULTATI}(b, a) = \{b_o : b_o &= \text{AGGIORNA}(\text{PREDIZIONE}(b, a), o) \text{ e} \\ o &\in \text{PERCEZIONI-POSSIBILI}(\text{PREDIZIONE}(b, a))\}. \end{aligned} \quad (4.5)$$

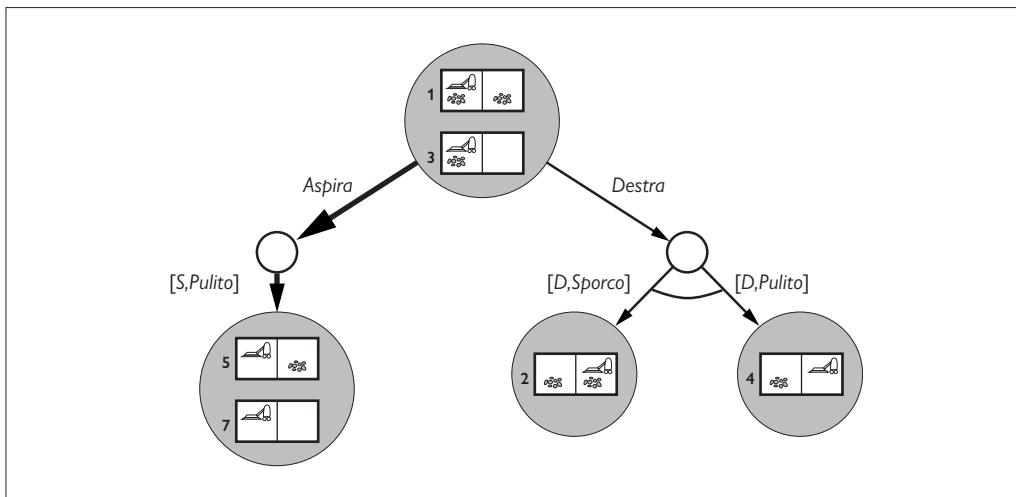


Figura 4.16
Il primo livello dell'albero di ricerca AND-OR per un problema nel mondo dell'aspirapolvere con sensori locali; *Aspira* è la prima azione della soluzione.

4.4.3 Risoluzione di problemi parzialmente osservabili

Nel paragrafo precedente abbiamo visto come derivare la funzione **RISULTATI** per un problema con stati-credenza non deterministico a partire da un problema fisico sottostante, data la funzione **PERCEZIONE**. Con questa formulazione, l'algoritmo di ricerca AND-OR della Figura 4.11 può essere applicato direttamente per ricavare una soluzione. La Figura 4.16 mostra una parte dell'albero di ricerca per il mondo dell'aspirapolvere a sensori locali, assumendo una percezione iniziale $[A, Sporco]$. La soluzione è il piano condizionale:

$[Aspira, Destra, \text{if } statoCredenza = \{6\} \text{ then } Aspira \text{ else []}]$.

Notate che, poiché abbiamo fornito all'algoritmo di ricerca AND-OR un problema di stato-credenza, viene restituito un piano condizionale che controlla lo stato-credenza anziché lo stato effettivo. È giusto così: in un ambiente parzialmente osservabile l'agente non sarà in grado di conoscere lo stato effettivo.

Come nel caso degli algoritmi di ricerca standard applicati a problemi senza sensori, l'algoritmo di ricerca AND-OR considera gli stati-credenza come scatole nere, alla pari di qualsiasi altro stato. Si potrebbe migliorare questo aspetto controllando se esistono stati-credenza generati in precedenza che sono sottoinsiemi o soprainsiemi dello stato corrente, come per i problemi senza sensori. Si potrebbero anche ricavare algoritmi di ricerca incrementali, analoghi a quelli descritti per i problemi senza sensori, che consentono una notevole velocizzazione rispetto all'approccio a scatola nera.

4.4.4 Un agente per ambienti parzialmente osservabili

Un agente per ambienti parzialmente osservabili formula un problema, invoca un algoritmo di ricerca (come **RICERCA-AND-OR**) per risolverlo ed esegue la soluzione. Ci sono due differenze principali fra questo agente e uno per ambienti completamente osservabili e deterministici: in primo luogo, la soluzione sarà un piano condizionale anziché una sequenza; se il primo passo è un'espressione if-then-else, l'agente dovrà verificare la condizione ed eseguire il ramo appropriato dell'istruzione condizionale. In secondo luogo, l'agente dovrà mantenere il proprio stato-credenza mentre esegue azioni e riceve percezioni. Questo processo assomiglia a quello di predizione-osservazione-aggiornamento dell'Equazione (4.5), ma in realtà è più semplice perché la percezione è data dall'ambiente e non calcolata dall'agente

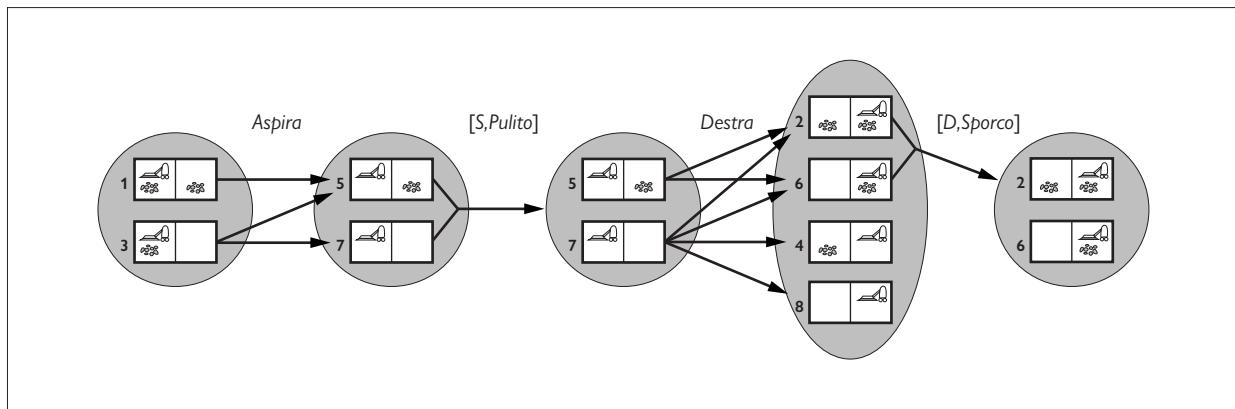


Figura 4.17 Due cicli predizione-aggiornamento di mantenimento dello stato-credenza nel mondo dell’aspirapolvere asilo con sensori locali.

stesso. Dato uno stato-credenza iniziale b , un’azione a e una percezione o , il nuovo stato-credenza è:

$$b' = \text{AGGIORNA}(\text{PREDIZIONE}(b, a), o). \quad (4.6)$$

Consideriamo un mondo dell’aspirapolvere *asilo* in cui gli agenti sono in grado di percepire soltanto lo stato del loro riquadro corrente, e ogni riquadro potrebbe diventare sporco in qualsiasi momento a meno che l’agente non sia attivamente impegnato nella pulizia in quel momento.⁷ La Figura 4.17 mostra lo stato-credenza mantenuto in questo ambiente.

In ambienti parzialmente osservabili, ovvero nella grande maggioranza degli ambienti del mondo reale, mantenere lo stato-credenza è una funzione centrale di qualsiasi sistema intelligente. Questa funzione è chiamata in vari modi, tra cui **monitoraggio**, **filtro** e **stima dello stato**. L’Equazione (4.6) è chiamata stimatore di stato *ricorsivo* perché calcola il nuovo stato-credenza sulla base del precedente anziché dell’intera sequenza di percezioni. Se l’agente non deve “rimanere indietro”, il calcolo deve essere veloce in modo da stare al passo con l’arrivo delle percezioni. Quando l’ambiente diventa più complesso, l’agente avrà soltanto il tempo di calcolare uno stato-credenza approssimato, eventualmente concentrandosi sulle conseguenze della percezione su aspetti dell’ambiente particolarmente interessanti al momento. La maggior parte dei lavori su questo problema è stata svolta per ambienti stocastici a stato continuo, con gli strumenti della teoria della probabilità, come vedremo nel Capitolo 14.

Nel seguito presentiamo un esempio in un ambiente discreto con sensori deterministici e azioni non deterministiche. Si tratta di un robot che ha un particolare compito di stima dello stato chiamato **localizzazione**: determinare dove si trova, data una mappa del mondo e una sequenza di percezioni e azioni. Il robot è posto nell’ambiente della Figura 4.18, simile a un labirinto, dispone di quattro sensori sonar che indicano se c’è un ostacolo (il muro esterno o un riquadro scuro nella figura) in ognuna delle quattro direzioni della bussola. La percezione è un vettore di bit, un bit per ciascuna delle direzioni nord, est, sud e ovest (in quest’ordine). Per esempio, 1011 significa che ci sono ostacoli a nord, sud e ovest, ma non est.

monitoraggio
filtro
stima dello stato

localizzazione

⁷ Le nostre scuse a chi non ha familiarità con l’effetto dei bambini piccoli sull’ambiente.

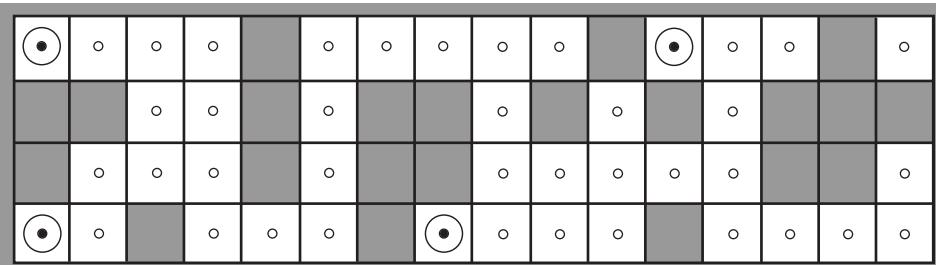
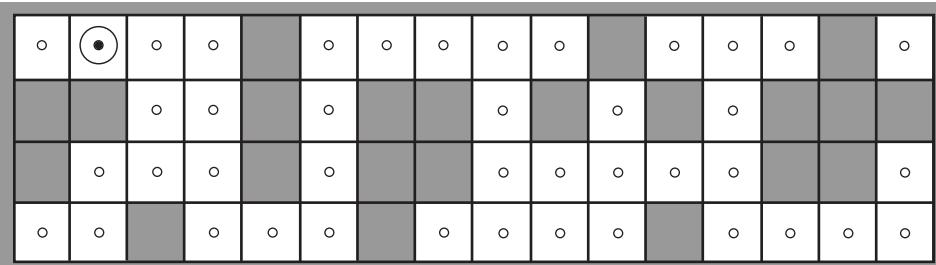
(a) Posizioni possibili del robot dopo $E_1 = 1011$ (b) Posizioni possibili del robot dopo $E_1 = 1011$ e $E_2 = 1010$

Figura 4.18 Posizioni possibili del robot, \odot , (a) dopo una osservazione $E_1 = 1011$ e (b) dopo una seconda osservazione $E_2 = 1010$. Quando i sensori sono privi di rumore e il modello di transizione è accurato, non vi sono altre posizioni possibili per il robot coerenti con questa sequenza di due osservazioni.

Ipotizziamo che i sensori forniscano dati perfettamente corretti e che il robot disponga di una mappa dell'ambiente corretta. Sfortunatamente, però, il sistema di locomozione del robot è guasto, perciò quando esegue un'azione *Destra*, si sposta a caso in uno dei riquadri adiacenti. Il compito del robot è di determinare la propria posizione attuale.

Supponiamo che il robot sia stato appena acceso e non sappia dove si trova – il suo stato-credenza iniziale b è costituito dall'insieme di tutte le posizioni. Poi il robot riceve la percezione 1011 ed effettua un aggiornamento utilizzando l'equazione $b_o = \text{AGGIORNA}(b1011)$, ottenendo le 4 posizioni mostrate nella Figura 4.18(a). Potete ispezionare il labirinto per vedere che queste sono le uniche quattro posizioni che originano la percezione 1011.

Poi il robot esegue un'azione *Destra*, ma il risultato è non deterministico. Il nuovo stato-credenza $b_a = \text{PREDIZIONE}(b_o, \text{Destra})$ contiene tutte le posizioni distanti un passo dalle posizioni in b_o . Quando arriva la seconda percezione, 1010, il robot esegue $\text{AGGIORNA}(b_a, 1010)$ e trova che lo stato-credenza si è ridotto all'unica posizione mostrata nella Figura 4.18(b). Questa è l'unica posizione che potrebbe essere il risultato di:

$$\text{AGGIORNA}(\text{PREDIZIONE}(\text{AGGIORNA}(b, 1011), \text{Destra}), 1010).$$

Con azioni non deterministiche il passo *PREDIZIONE* ingrandisce lo stato-credenza, ma il passo *AGGIORNA* lo riduce, purché le percezioni forniscano informazioni di identificazione utili. A volte le percezioni non aiutano molto nella localizzazione: se vi fossero uno o più lunghi corridoi est-ovest, un robot potrebbe ricevere una lunga sequenza di percezioni 1010, senza mai sapere in che punto del corridoio (o dei corridoi) si trovi. Ma per ambienti con variazione ragionevole della geografia, la localizzazione spesso converge rapidamente a un punto singolo, anche quando le azioni sono non deterministiche.

Che cosa accade se i sensori sono guasti? Se possiamo ragionare soltanto in logica booleana, dobbiamo considerare ogni bit di sensore corretto oppure errato, che significa non avere alcuna informazione percettiva. Ma vedremo che il ragionamento probabilistico (Capitolo 13) consente di estrarre informazioni utili anche da un sensore guasto, purché questo si sbagli per meno di metà del tempo.

4.5 Agenti per ricerca online e ambienti sconosciuti

ricerca offline
ricerca online

Fin qui ci siamo concentrati su agenti che utilizzano algoritmi di **ricerca offline**, calcolando una soluzione completa prima di effettuare la prima azione. Al contrario, nella **ricerca online**⁸ un agente opera alternando computazione e azione: prima esegue un’azione, poi osserva l’ambiente e determina l’azione successiva. La ricerca online si presta bene agli ambienti dinamici o semidinamici, nei quali non c’è tempo per stare immobili a calcolare l’azione successiva. Inoltre, la ricerca online è utile anche in domini non deterministici, perché consente all’agente di concentrare le attività di calcolo sulle contingenze che si verificano effettivamente, anziché su quelle che *potrebbero* verificarsi ma probabilmente non lo faranno.

Naturalmente c’è un compromesso: più un agente pianifica in anticipo, meno spesso gli capiterà di trovarsi nei guai. In ambienti ignoti, l’agente non conosce gli stati e gli effetti delle azioni e deve sfruttare le sue azioni come esperimenti per poter apprendere le caratteristiche dell’ambiente.

problema di costruzione di mappe

Un esempio classico di ricerca online è il **problema di costruzione di mappe**: un robot che si trova in un edificio sconosciuto e deve esplorarlo per costruire una mappa che in seguito possa essere usata per andare da *A* a *B*. I metodi per uscire dai labirinti (una conoscenza necessaria per gli aspiranti eroi dei tempi antichi) rappresentano anch’essi esempi di algoritmi di ricerca online. Quella spaziale, comunque, non è l’unica forma di esplorazione. Considerate un neonato: può eseguire molte azioni, ma non conosce il risultato di nessuna, e ha sperimentato solo pochi dei possibili stati raggiungibili.

4.5.1 Problemi di ricerca online

Un problema di ricerca online viene risolto con attività di elaborazione, percezione e azione. Iniziamo supponendo che l’ambiente sia deterministico e completamente osservabile (nel Capitolo 17 queste ipotesi sono rilassate) e stabiliamo che l’agente conosca solo quanto segue:

- **AZIONI(*s*)**, le azioni permesse nello stato *s*;
- $c(s, a, s')$, il costo di applicare l’azione *a* nello stato *s* per arrivare nello stato *s'*. Notate che non può essere usato finché l’agente non appura che *s'* è il risultato dell’azione;
- **È-OBIETTIVO(*s*)**, il test obiettivo.

Osservate in particolare che l’agente *non può* determinare **RISULTATO(*s,a*)** se non trovandosi effettivamente in *s* ed eseguendo *a*. Per esempio, nel problema di labirinto mostrato nella Figura 4.19, l’agente non sa che muoversi verso l’*Alto* da (1,1) porta a (1,2); e neppure sa, dopo aver compiuto quell’azione, che andare in *Basso* lo riporterà indietro in (1,1). In

⁸ Il termine “online” qui si riferisce ad algoritmi che devono elaborare i dati in input nel momento in cui sono ricevuti, invece di aspettare di averli tutti disponibili. In questo senso, il termine non è correlato al concetto di “essere connessi a Internet”.

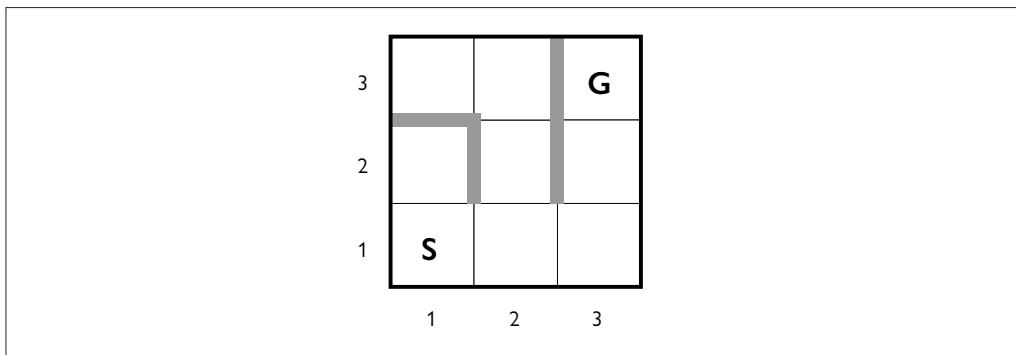


Figura 4.19
Un semplice problema di labirinto. L'agente parte da S e deve arrivare in G, ma non sa nulla dell'ambiente.

alcune applicazioni questo grado di ignoranza può essere ridotto: per esempio, un robot esploratore potrebbe conoscere il funzionamento dei propri movimenti e ignorare solo la posizione degli ostacoli.

Infine, l'agente potrebbe avere accesso a una funzione euristica ammmissible $h(s)$ capace di stimare la distanza dallo stato corrente a uno stato obiettivo. Per esempio, nella Figura 4.19, l'agente potrebbe conoscere la posizione dell'obiettivo ed essere capace di usare come euristica la distanza Manhattan (cfr. Paragrafo 3.6).

Generalmente lo scopo dell'agente è raggiungere uno stato obiettivo minimizzando il costo (un altro possibile scopo può essere semplicemente di esplorare l'intero ambiente). Il costo è rappresentato da quello totale del cammino effettivamente percorso dall'agente. È prassi comune confrontare questo costo con quello del cammino che l'agente potrebbe seguire se conoscesse in anticipo lo spazio di ricerca, cioè il cammino ottimo nell'ambiente noto. Nel linguaggio degli algoritmi online, questo confronto determina il cosiddetto **rappporto di competitività** (*competitive ratio*), che auspiciamo sia il più piccolo possibile.

Gli esploratori online sono vulnerabili ai **vicoli ciechi**, stati da cui non è raggiungibile alcuno stato obiettivo. Se l'agente non conosce il significato di ogni azione, potrebbe eseguire l'azione “salta nella fossa più profonda” e quindi non raggiungere mai l'obiettivo. In generale, *nessun algoritmo può evitare i vicoli ciechi in tutti gli spazi degli stati*. Consideriamo i due spazi degli stati nella Figura 4.20(a), ognuno dei quali ha un vicolo cieco. Un algoritmo di ricerca online che ha visitato gli stati S e A non è in grado di sapere se si trova nello stato superiore o in quello inferiore, poiché i due appaiono identici ai suoi occhi. Quindi non vi è modo per cui l'agente possa sapere come scegliere l'azione corretta in entrambi gli spazi degli stati. Questo è un esempio di quello che gli anglosassoni chiamano **adversary argument** (letteralmente “argomentazione dell'avversario”): possiamo immaginare un “avversario” che costruisce lo spazio degli stati mentre l'agente lo esplora, posizionando gli obiettivi e i vicoli ciechi dove gli pare, come nella Figura 4.20(b).

I vicoli ciechi costituiscono una difficoltà reale per l'esplorazione robotica: scale, rampe, scarpate, strade a senso unico e anche il terreno naturale presentano stati da cui alcune azioni sono **irreversibili**: non vi è modo di tornare allo stato precedente. L'algoritmo di esplorazione che presentiamo nel seguito può funzionare soltanto in spazi degli stati **esplorabili in modo sicuro**, ovvero dove si possa sempre arrivare a uno stato obiettivo da ogni stato raggiungibile. Gli spazi degli stati con azioni reversibili, come i labirinti e i rompicapi a 8 tasselli, sono chiaramente esplorabili in modo sicuro (se hanno una soluzione). Tratteremo più in dettaglio il concetto dell'esplorazione sicura nel Paragrafo 22.3.2 del Volume 2.

Anche negli ambienti esplorabili in modo sicuro non può essere garantito un rapporto di competitività limitato. È facile dimostrarlo in ambienti con azioni irreversibili, ma in effetti ciò rimane vero anche nel caso reversibile, come si vede nella Figura 4.20(b). Per questa ragione, spesso si caratterizzano le prestazioni degli algoritmi di ricerca online in rapporto alle

rapporto di competitività
vicolo cieco



adversary argument

azioni irreversibili
stati esplorabili in modo sicuro

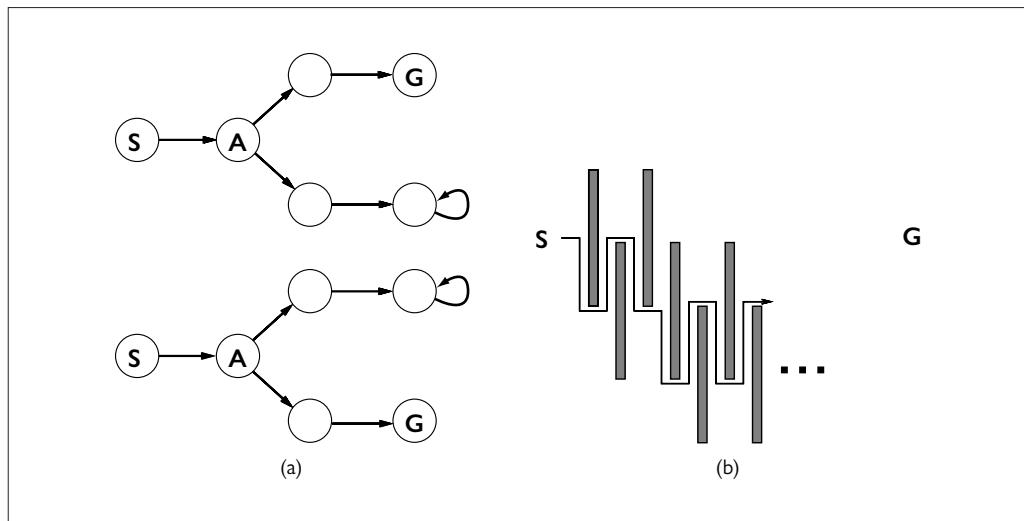


Figura 4.20 (a) Due spazi degli stati che potrebbero condurre un agente per ricerca online in un vicolo cieco. Un qualsiasi agente dato fallirà in almeno uno di questi spazi. (b) Un ambiente bidimensionale che può obbligare un agente per ricerca online a seguire un cammino verso l'obiettivo arbitrariamente inefficiente. Qualsiasi scelta faccia l'agente, l'avversario blocca il cammino corrispondente con un lungo muro sottile, in modo tale che il cammino finale sia molto più lungo del minimo teorico.

dimensioni dell'intero spazio degli stati e non rispetto alla sola profondità dell'obiettivo più vicino alla radice.

4.5.2 Agenti per ricerca online

Dopo ogni azione, un agente online in un ambiente osservabile riceve una percezione che gli comunica lo stato raggiunto; in base a questa esso può arricchire la sua mappa dell'ambiente. La mappa aggiornata viene poi utilizzata per pianificare l'azione successiva: quest'alternanza tra pianificazione e azione fa sì che gli algoritmi di ricerca online siano molto diversi da quelli offline che abbiamo considerato finora: gli algoritmi offline esplorano il loro *modello* dello spazio degli stati, mentre gli algoritmi online esplorano il mondo reale. Per esempio, l'algoritmo A^* può espandere un nodo in una parte dello spazio e poi subito dopo un altro nodo in un'altra parte distante, perché l'espansione dei nodi viene effettuata mediante azioni simulate e non reali. Un algoritmo online, invece, può scoprire successori solamente per uno stato che sta occupando fisicamente. Per evitare di spostarsi in uno stato distante per espandere il nodo successivo, sembra più sensato eseguire le espansioni seguendo un ordine *locale*. La ricerca in profondità ha esattamente questa proprietà, perché (tranne quando l'algoritmo torna sui propri passi, cioè fa backtracking) il nodo successivo è sempre figlio di quello appena espanso.

La Figura 4.21 presenta un agente per l'esplorazione online in profondità (per azioni deterministiche ma ignote). Questo agente memorizza la sua mappa in una tabella, *risultato*[s, a], che registra lo stato risultante dall'esecuzione dell'azione a nello stato s (per azioni non deterministiche, l'agente potrebbe registrare un insieme di stati in *risultati*[s, a]). Ogni volta che nello stato corrente ci sono azioni ancora inesplorate, l'agente ne prova una. Le difficoltà sorgono quando l'agente ha provato tutte le azioni in uno stato: nella ricerca offline, lo stato è semplicemente cancellato dalla coda; in quella online, l'agente deve tornare fisicamente sui propri passi nel mondo fisico. Per la ricerca in profondità, questo significa tornare nello stato da cui l'agente è entrato (più di recente) in quello corrente. Per far questo l'algoritmo mantiene in memoria un'altra tabella che elenca, per ogni stato, gli stati predecessori a cui

```

function AGENTE-ONLINE-RIP(problema, s') returns un'azione
    s, a, stato precedente e azione, inizialmente null
    persistent: risultato, una tabella che mappa (s, a) su s', inizialmente vuota
                  nonprovate, una tabella che mappa s su una lista di azioni non ancora provate
                  nonbacktrack, una tabella che mappa s su una lista di stati a cui non si è tornati

    if problema.È-OBIETTIVO(s') then return stop
    if s' è un nuovo stato (non in nonprovate) then nonprovate[s']  $\leftarrow$  problema.AZIONI(s')
    if s non è null then
        risultato[s, a]  $\leftarrow$  s'
        aggiungi s nella prima posizione di [s']
    if nonprovate[s'] è vuoto then
        if nonbacktrack[s'] è vuoto then return stop
        else a  $\leftarrow$  un'azione b tale che risultato[s', b] = POP(nonbacktrack[s'])
    else a  $\leftarrow$  POP(nonprovate[s'])
    s  $\leftarrow$  s'
    return a

```

Figura 4.21 Un agente per ricerca online che utilizza un'esplorazione in profondità. L'agente può esplorare in modo sicuro solo spazi degli stati in cui ogni azione può essere “annullata” da un'altra.

l'agente non è ancora ritornato. Se l'agente non ha più stati a cui ritornare, la sua ricerca è completa.

Raccomandiamo ai lettori di ricostruire su carta il progresso di AGENTE-ONLINE-RIP applicato al labirinto della Figura 4.19. È abbastanza facile vedere che l'agente, nel caso peggiore, attraverserà ogni collegamento nello spazio degli stati esattamente due volte. Per un'esplorazione, questo risultato è ottimo; per trovare un obiettivo, d'altra parte, il rapporto di competitività potrebbe risultare arbitrariamente alto se l'agente “se ne va a spasso” per una lunga escursione quando c'è uno stato obiettivo proprio accanto a quello iniziale. Una variante online della ricerca ad approfondimento iterativo risolve questo problema; se l'ambiente è un albero uniforme, il rapporto di competitività di un agente così fatto sarà una piccola costante.

Per via del suo uso del backtracking, AGENTE-ONLINE-RIP funziona solo negli spazi degli stati dove le azioni sono reversibili. Ci sono algoritmi leggermente più complessi che possono operare in tutti gli spazi degli stati, ma nessuno di essi ha un rapporto di competitività limitato.

4.5.3 Ricerca locale online

Come quella in profondità, anche la **ricerca hill climbing** gode della proprietà di località nelle espansioni dei suoi nodi. In effetti, dato che tiene in memoria solamente lo stato corrente, la ricerca hill climbing è già un algoritmo online! Sfortunatamente, l'algoritmo di base non è molto utile per l'esplorazione, perché l'agente si può bloccare in corrispondenza di un massimo locale. Per di più, non si può neppure usare il meccanismo del riavvio casuale, perché l'agente non può teletrasportarsi in un nuovo stato iniziale.

Invece dei riavvii casuali, per esplorare l'ambiente si potrebbe considerare l'uso di un **random walk** (letteralmente, “passeggiata casuale”), che sceglie semplicemente a caso una delle azioni possibili nello stato corrente; è possibile prediligere quelle che non sono ancora

random walk

Figura 4.22

Un ambiente in cui un random walk richiederà un numero di passi esponenzialmente grande per raggiungere l'obiettivo G.

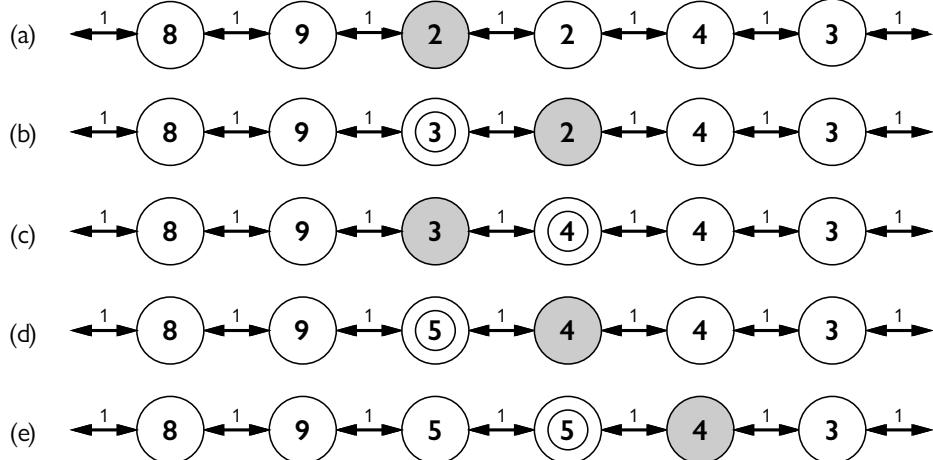
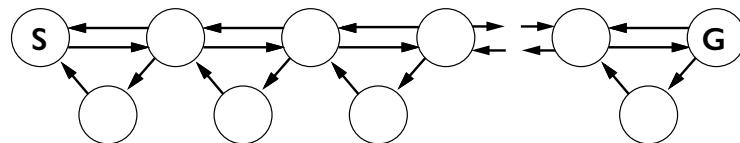


Figura 4.23 Cinque iterazioni di LRTA* in uno spazio degli stati monodimensionale. Ogni stato è etichettato con $H(s)$, la stima corrente del costo per raggiungere un obiettivo, e ogni arco ha un costo di azione pari a 1. Lo stato ombreggiato indica la posizione dell'agente, e a ogni iterazione le stime di costo aggiornate sono evidenziate con un cerchio doppio.

state provate. È facile dimostrare che una passeggiata casuale *prima o poi* troverà un obiettivo o completerà la sua esplorazione, a patto che lo spazio degli stati sia finito ed esplorabile in modo sicuro.⁹ D'altra parte, il processo può essere molto lento. La Figura 4.22 mostra un ambiente in cui un random walk richiederà un numero esponenziale di passi per raggiungere l'obiettivo, perché per ogni stato nella riga superiore a eccezione di S, la probabilità di tornare indietro è doppia rispetto a quella di avanzare. Naturalmente l'esempio è fittizio, ma ci sono molti spazi degli stati nel mondo reale la cui topologia causa simili “trappole” per i random walk.

Arricchire la ricerca hill climbing con *memoria* anziché casualità si rivela un approccio più efficace. L'idea è memorizzare la “miglior stima corrente” $H(s)$ del costo per raggiungere l'obiettivo da ogni stato visitato. $H(s)$ all'inizio non è altro che la stima euristica $h(s)$, ma viene aggiornata man mano che l'agente acquisisce esperienza nello spazio degli stati.

La Figura 4.23 mostra un semplice esempio in uno spazio monodimensionale. In (a), l'agente sembra bloccato in un minimo locale piatto, nello stato ombreggiato. Invece di re-

⁹ Le passeggiate casuali sono complete su griglie infinite mono- e bidimensionali. Su una griglia tridimensionale, la probabilità che il cammino ritorni in un qualche momento al punto iniziale è solo di circa 0,3405 (Hughes, 1995).

```

function AGENTE-LRTA*(problema, s', h) returns un'azione
    s, a, rispettivamente stato e azione precedenti, inizialmente null
    persistent: risultato, una tabella che mappa (s, a) su s', inizialmente vuota
    H, una tabella che mappa s su una stima di costo, inizialmente vuota

    if È-OBIETTIVO(s') then return stop
    if s' è un nuovo stato (non in H) then H[s']  $\leftarrow h(s')$ 
    if s non è null then
        risultato[s, a]  $\leftarrow s'$ 
        H[s]  $\leftarrow \min_{b \in \text{problema.AZIONI}(s)} \text{COSTO-LRTA}^*(\text{problema}, s, b, \text{risultato}[s, b], H)$ 
        a  $\leftarrow \operatorname{argmin}_{b \in \text{problema.AZIONI}(s)} \text{COSTO-LRTA}^*(\text{problema}, s', b, \text{risultato}[s', b], H)$ 
    return a

function COSTO-LRTA*(problema, s, a, s', H) returns una stima di costo
    if s' è indefinito then return h(s)
    else return problema.COSTO-AZIONE(s, a, s') + H[s']

```

Figura 4.24 Agente-LRTA* seleziona un'azione in base ai valori degli stati adiacenti, che vengono aggiornati mentre l'agente si muove nello spazio degli stati.

stare lì, l'agente deve seguire quello che sembra il cammino migliore verso l'obiettivo in base alle stime correnti di costo per i suoi vicini. Il costo stimato per arrivare all'obiettivo passando per un vicino *s'* corrisponde al costo per spostarsi in *s'* sommato a quello stimato per muoversi da lì all'obiettivo, cioè $c(s, a, s') + H(s')$. Nell'esempio, le due azioni possibili hanno costi stimati 1 + 9 a sinistra e 1 + 2 a destra, quindi la scelta migliore sembra andare a destra.

In (b) è chiaro che il costo stimato di 2 per lo stato ombreggiato in (a) era esageratamente ottimistico. Dato che partendo da quella posizione la mossa migliore costa 1 e porta a uno stato che si trova ad almeno 2 passi dall'obiettivo, lo stato ombreggiato dev'essere almeno 3 passi distante dall'obiettivo e il suo valore *H* dev'essere aggiornato di conseguenza, come si vede nella Figura 4.23(b). Continuando questo processo, l'agente si muoverà ancora due volte avanti e indietro, aggiornando *H* ogni volta e “appiattendo” il minimo locale fino a sfuggirne sulla destra.

La Figura 4.24 riporta un agente che implementa questo schema, che prende il nome di “A* con apprendimento in tempo reale” (indicato con **LRTA***, acronimo di *learning real-time A**). Come AGENTE-ONLINE-RIP, anche questo agente costruisce una mappa dell'ambiente nella tabella *risultato*. L'algoritmo aggiorna la stima del costo dello stato che ha appena lasciato e poi sceglie la mossa “apparentemente migliore” in base alle stime correnti dei costi. Un aspetto importante è che si pensa sempre che le azioni che non sono ancora state provate in uno stato *s* conducano immediatamente all'obiettivo con il minimo costo possibile, cioè *h(s)*. Questo **ottimismo in condizioni di incertezza** incoraggia l'agente a esplorare nuovi cammini potenzialmente promettenti.

Un agente LRTA* troverà sempre un obiettivo, a patto che l'ambiente sia finito ed esplorabile in modo sicuro. A differenza di A*, comunque, l'algoritmo non è completo in spazi degli stati infiniti: in alcuni casi può lasciarsi sviare senza possibilità di ritorno. Dato un ambiente di *n* stati, nel caso pessimo lo può esplorare in $O(n^2)$ passi, ma spesso si comporta molto meglio. LRTA* appartiene a una grande famiglia di agenti online che possono essere

LRTA*

**ottimismo
in condizioni
di incertezza**

definiti specificando in modi diversi la regola di selezione delle azioni e quella di aggiornamento delle stime; tratteremo questa famiglia nel Capitolo 22 del Volume 2.

4.5.4 Apprendimento nella ricerca online

Lo stato iniziale di ignoranza degli agenti per ricerca online fornisce diverse opportunità per l'apprendimento. Prima di tutto un agente deve imparare la “mappa” dell’ambiente (o, più precisamente, le conseguenze di ogni azione in ogni stato) semplicemente memorizzando le sue esperienze. In un secondo momento gli agenti che effettuano ricerche locali devono acquisire stime più accurate del valore di ogni stato utilizzando adeguate regole di aggiornamento, come abbiamo visto per LRTA*. Nel Capitolo 22 del Volume 2 vedremo che questi aggiornamenti a un certo punto convergono ai valori *esatti* di ogni stato, a patto che l’agente esplori lo spazio degli stati nel modo corretto. Una volta noti i valori esatti, si possono prendere decisioni ottime semplicemente muovendosi verso il successore di costo minore: in altre parole, la strategia di hill climbing puro diventa ottima.

Se avete seguito il nostro suggerimento di seguire passo passo il comportamento di AGENTE-ONLINE-RIP nell’ambiente della Figura 4.19, avrete notato che non è particolarmente sagace. Per esempio, dopo aver visto che l’azione *Alto* passa da (1,1) a (1,2), l’agente continua a non sapere che l’azione *Basso* torna indietro in (1,1), o che la stessa azione *Alto* serve anche a passare da (2,1) a (2,2), da (2,2) a (2,3) eccetera. In generale, ci piacerebbe che l’agente imparasse che *Alto* aumenta il valore della coordinata *y* (a meno che non ci sia un ostacolo), che *Basso* lo diminuisce e così via.

Affinché questo accada, servono due elementi. Prima di tutto è necessaria una rappresentazione formale ed esplicitamente manipolabile di questo tipo di regole generali; finora infatti quest’informazione è rimasta chiusa in una scatola nera chiamata funzione RISULTATO. I Capitoli da 8 a 11 sono dedicati a quest’argomento. Inoltre, ci serviranno algoritmi che possono costruire regole generali utili partendo dalle specifiche osservazioni dell’agente: questi saranno trattati nel Capitolo 19 del Volume 2.

Se pensiamo che in futuro potremmo essere chiamati a risolvere parecchi problemi di questo tipo, vale la pena investire tempo (e memoria) per rendere più semplici tali ricerche. Esistono diversi modi per farlo, tutti generalmente ricompresi nella **ricerca incrementale**. Potremmo mantenere in memoria l’albero di ricerca e riutilizzarne le parti invariate nel nuovo problema. Potremmo mantenere i valori *h* dell’euristica e aggiornarli quando disponiamo di nuove informazioni, perché il mondo è cambiato o perché abbiamo calcolato stime migliori. Oppure potremmo mantenere i valori *g* del cammino migliore, usandoli per comporre una nuova soluzione e aggiornandoli quando il mondo cambia.

**ricerca
incrementale**

4.6 Riepilogo

Questo capitolo ha preso in esame gli algoritmi di ricerca per problemi in ambienti parzialmente osservabili, non deterministici, ignoti e continui.

- I metodi di *ricerca locale* come **hill climbing** mantengono in memoria solo un piccolo numero di stati. Sono stati applicati in problemi di ottimizzazione, in cui il concetto è quello di trovare uno stato ad alto punteggio senza preoccuparsi del cammino per raggiungerlo. Sono stati sviluppati diversi algoritmi stocastici, tra cui il **simulated annealing**, che fornisce soluzioni ottime quando viene impostata una “velocità di raffreddamento” appropriata.
- Si possono usare numerosi metodi di ricerca locale anche per risolvere problemi negli spazi continui. I problemi di **programmazione lineare** e di **ottimizzazione convessa** obbediscono a determinati vincoli sulla forma dello spazio degli stati e sulla natura della funzione obiettivo, e ammettono algoritmi con complessità temporale polinomiale che spesso sono estremamente efficienti nella pratica. Per alcuni problemi matematicamente ben

formati, possiamo trovare il massimo usando l'analisi matematica per determinare dove il gradiente è zero; per altri problemi dobbiamo farlo con il gradiente empirico, che misura la differenza della fitness tra due punti vicini.

- Un **algoritmo evolutivo** è una ricerca hill climbing stocastica in cui viene tenuta in memoria una popolazione di stati. I nuovi stati sono generati con la **mutazione** e il **crossover**, che combina coppie di stati.
- In **ambienti non deterministici**, gli agenti possono applicare la ricerca AND-OR per generare **piani condizionali** che raggiungono l'obiettivo a prescindere da quali risultati si verifichino durante l'esecuzione.
- Quando l'ambiente è parzialmente osservabile, lo **stato-credenza** rappresenta l'insieme dei possibili stati in cui l'agente potrebbe trovarsi.
- Gli algoritmi di ricerca standard possono essere applicati direttamente allo spazio degli stati-credenza per risolvere **problemI senza sensori**, e una ricerca AND-OR in questo spazio può risolvere problemI parzialmente osservabili generali. Gli algoritmi incrementali che costruiscono soluzioni stato per stato all'interno di uno stato-credenza sono spesso più efficienti.
- I **problemI di esplorazione** sorgono quando un agente non conosce gli stati e le azioni del suo ambiente. Se l'ambiente è esplorabile in modo sicuro, gli agenti possono svolgere una **ricerca online** per costruire una mappa e trovare un obiettivo, se esiste. Per sfuggire ai minimi locali un metodo efficace è aggiornare le stime euristiche in base all'esperienza.

Note storiche e bibliografiche

Le tecniche di ricerca locale hanno una lunga storia nella matematica e nell'informatica. In effetti, il metodo Newton–Raphson (Newton, 1671; Raphson, 1690) può essere considerato un metodo di ricerca locale molto efficiente per spazi continui in cui è disponibile informazione sul gradiente. Brent (1973) è un riferimento classico per gli algoritmi di ottimizzazione che non richiedono tale informazione. La ricerca beam, che abbiamo presentato come un algoritmo di ricerca locale, è nata come una variante ad ampiezza limitata della programmazione dinamica per il riconoscimento vocale nel sistema HARPY (Lowerre, 1976). Un algoritmo simile è analizzato in profondità da Pearl (1984, Cap. 5).

In anni recenti la ricerca locale è stata rinvigorita da risultati sorprendentemente buoni per grandi problemI di soddisfacimento di vincoli come quelli delle *n* regine (Minton *et al.*, 1992) e della soddisfacibilità booleana (Selman *et al.*, 1992) e grazie anche all'apporto della causalità, dello svolgimento di ricerche multiple simultanee e di altre migliorie. La rinascita di quelli che Christos Papadimitriou ha chiamato “algoritmi New Age” ha suscitato interesse tra gli informatici teorici (Koutsoupias e Papadimitriou, 1992; Al-dous e Vazirani, 1994). Nel campo della ricerca

operativa, ha guadagnato popolarità una variante dell'hill climbing chiamata **ricerca tabù** (Glover e Laguna, 1997). Questo algoritmo mantiene una “lista tabù” di *k* stati precedentemente visitati a cui non si può ritornare; oltre a migliorare l'efficienza delle ricerche su grafo, questa lista può permettere all'algoritmo di evitare alcuni minimi locali.

Un'altra utile miglioria dell'hill climbing è l'algoritmo STAGE (Boyan e Moore, 1998), che usa i massimi locali trovati mediante l'hill climbing a riavvio casuale per farsi un'idea della forma generale del panorama. L'algoritmo adatta una superficie quadratica “liscia” a un insieme di massimi locali e poi calcola analiticamente il massimo globale della superficie stessa: questo diventa il nuovo punto di partenza della ricerca. (Gomes *et al.*, 1998) hanno dimostrato che i tempi di esecuzione degli algoritmi che effettuano sistematicamente il backtracking spesso hanno una **distribuzione a coda pesante**, che significa che la probabilità di avere un tempo di esecuzione molto lungo è maggiore di quella che si potrebbe prevedere se i tempi di esecuzione avessero una distribuzione esponenziale. Quando la distribuzione del tempo di esecuzione è a coda pesante, la tecnica dei riavvii casuali trova

più velocemente una soluzione, in media, rispetto a una singola esecuzione portata fino al completamento. Hoos e Stützle (2004) hanno dedicato un intero libro a questo argomento.

Il simulated annealing fu descritto per la prima volta da Kirkpatrick *et al.* (1983), che si ispirarono direttamente all'**algoritmo Metropolis** usato per simulare complessi sistemi fisici (Metropolis *et al.*, 1953) e che fu inventato, si dice, durante una cena sociale a Los Alamos. Il simulated annealing ora è un campo di ricerca a sé stante, su cui vengono pubblicati ogni anno centinaia di articoli.

La ricerca di soluzioni ottime negli spazi continui è l'argomento di molte discipline, tra cui la **teoria dell'ottimizzazione**, la **teoria del controllo ottimo** e il **calcolo delle variazioni**. Le tecniche di base sono spiegate bene da Bishop (1995); Press *et al.* (2007) tratta un'ampia varietà di algoritmi e fornisce strumenti software.

Gli studiosi di algoritmi di ricerca e ottimizzazione hanno tratto ispirazione da un'ampia varietà di campi di studio: metallurgia (simulated annealing), biologia (algoritmi genetici), neuroscienze (reti neurali), montagna (hill climbing), economia (algoritmi basati sul mercato (Dias *et al.*, 2006)), fisica (sciami di particelle (Li e Yao, 2012) e vetri di spin (Mézard *et al.*, 1987)); comportamento animale (apprendimento per rinforzo, ottimizzatori del “lupo grigio” (Mirjalili e Lewis, 2014)); ornitologia (ricerca del cuculo (Yang e Deb, 2014)); entomologia (ottimizzazione basata su colonie di formiche (Dorigo *et al.*, 2008), colonie di api (Karaboga e Basturk, 2007), lucciole (Yang, 2009) e vermi luminosi (Krishnanand e Ghose, 2009)); e altri.

La **programmazione lineare** (PL) fu studiata in modo sistematico per primo dal matematico Leonid Kantorovich (1939). È stata una delle prime applicazioni dei computer; l'algoritmo del simplesso (Dantzig, 1949) è ancora in uso, nonostante abbia complessità esponenziale nel caso peggiore. Karmarkar (1984) ha sviluppato i metodi di **punto interno**, molto più efficienti, per i quali si è dimostrata complessità polinomiale per la classe più generale dei problemi di ottimizzazione convessa da parte di Nesterov e Nemirovski (1994). Eccellenti introduzioni all'ottimizzazione convessa sono fornite da Ben-Tal e Nemirovski (2001) e da Boyd e Vandenberghe (2004).

Il lavoro di Sewall Wright (1931) sul concetto di **panorama della fitness** è stato un importante precursore dello sviluppo degli algoritmi genetici. Negli anni 1950 molti studiosi di statistica, tra cui Box (1957) e Friedman (1959), applicarono tecniche evolutive per risolvere problemi di ottimizzazione: l'approccio tuttavia

non divenne popolare finché Rechenberg (1965) introdusse **strategie evolutive** per risolvere problemi di ottimizzazione applicati ai profili alari. Negli anni 1960 e 1970 John Holland (1975) propugnò l'uso degli algoritmi genetici, sia come utile strumento di ottimizzazione che come metodo per espandere la nostra comprensione dell'adattamento (Holland, 1995).

Il movimento della **vita artificiale** (Langton, 1995) ha portato quest'idea un passo oltre, considerando i prodotti degli algoritmi genetici come *organismi* anziché soluzioni di problemi. L'effetto Baldwin discusso in questo capitolo è stato proposto in modo più o meno simultaneo da Conwy Lloyd Morgan (1896) e James Baldwin (1896). Le simulazioni al computer hanno aiutato a chiarirne le implicazioni (Hinton e Nowlan, 1987; Ackley e Littman, 1991; Morgan e Griffiths, 2015). Smith e Szathmáry (1999), Ridley (2004) e Carrol (2007) discutono le fondamenta generali dell'evoluzione.

La maggior parte dei confronti tra gli algoritmi genetici e gli altri approcci (in particolare l'hill climbing stocastico) hanno riscontrato che i primi convergono più lentamente (O'Reilly e Oppacher, 1994; Mitchell *et al.*, 1996; Juels e Wattenberg, 1996; Baluja, 1997). Comprensibilmente questi risultati non sono popolari nella comunità degli algoritmi genetici, ma i recenti tentativi di questa comunità di formulare la ricerca basata su popolazioni come una forma approssimata di apprendimento bayesiano potrebbero aiutare a colmare la distanza tra gli algoritmi genetici e i loro critici (Pelikan *et al.*, 1999). La teoria dei **sistemi dinamici quadratici** potrebbe anche spiegare le prestazioni degli algoritmi genetici (Rabani *et al.*, 1998). Ci sono applicazioni pratiche degli algoritmi genetici davvero impressionanti, in aree diverse quali la progettazione di un'antenna (Lohn *et al.*, 2001), il CAD o Computer-Aided Design (Renner ed Ekart, 2003), i modelli climatici (Stanislawska *et al.*, 2015), la medicina (Ghereri *et al.*, 2015) e la progettazione di reti neurali deep (Miikkulainen *et al.*, 2019).

La **programmazione genetica** è un sottocampo degli algoritmi genetici in cui le rappresentazioni sono programmi e non stringhe di bit. I programmi sono rappresentati sotto forma di alberi sintattici, in un linguaggio di programmazione standard o in formati specifici per rappresentare circuiti, controller di robot e così via. Il crossover comporta la suddivisione in sottoalberi in modo tale da garantire che i discendenti siano costituiti da espressioni ben formate.

L'interesse nella programmazione genetica è stato ravvivato dal lavoro di John Koza (1992, 1994), ma la

tecnica risale almeno ai primi esperimenti sul codice macchina di Friedberg (1958) e sugli automi a stati finiti da parte di Fogel *et al.* (1966). Come per gli algoritmi genetici, sull'effettiva efficacia della tecnica c'è un certo dibattito. Koza *et al.* (1999) descrivono esperimenti nell'uso della programmazione genetica per progettare circuiti.

Le riviste *Evolutionary Computation* e *IEEE Transactions on Evolutionary Computation* trattano gli algoritmi evolutivi; si possono trovare articoli anche in *Complex Systems*, *Adaptive Behavior* e *Artificial Life*. Il congresso più importante è la *Genetic and Evolutionary Computation Conference (GECCO)*. I libri di Mitchell (1996), Fogel (2000), Langdon e Poli (2002), Poli *et al.* (2008) offrono buone panoramiche sull'argomento.

La impredictibilità e la parziale osservabilità degli ambienti reali furono riconosciuti ben presto nei progetti di robotica che utilizzavano tecniche di pianificazione, tra cui Shakey (Fikes *et al.*, 1972) e FREDDY (Michie, 1972). I problemi ricevettero maggiore attenzione dopo la pubblicazione dell'importante articolo di McDermott (1978a), *Planning and Acting*.

Il primo lavoro che utilizzò esplicitamente alberi AND-OR sembra essere stato il programma SAINT di Slagle per l'integrazione simbolica, citato nel Capitolo 1. Amarel (1967) applicò l'idea alla dimostrazione di teoremi nella logica proposizionale, tema discusso nel Capitolo 7, e presentò un algoritmo di ricerca simile a RICERCA-AND-OR. L'algoritmo fu ulteriormente sviluppato da Nilsson (1971), che descrisse anche AO* che, come suggerisce il nome, trova soluzioni ottime. AO* fu ulteriormente migliorato da Martelli e Montanari (1973).

AO* è un algoritmo top-down; una generalizzazione bottom-up di A* è A*LD, che sta per A* Lightest Derivation (Felzenszwalb e McAllester, 2007). L'interesse nella ricerca AND-OR si è rinnovato nei primi anni 2000, con nuovi algoritmi per trovare soluzioni cicliche (Jimenez e Torras, 2000; Hansen e Zilberman, 2001) e nuove tecniche ispirate dalla programmazione dinamica (Bonet e Geffner, 2005).

L'idea di trasformare problemi parzialmente osservabili in problemi di stato-credenza è nata con Astrom (1965) per il caso molto più complesso dell'incertezza probabilistica (cfr. Capitolo 17). Erdmann e Mason (1988) studiarono il problema della manipolazione robotica senza sensori, utilizzando una forma continua di ricerca nello spazio degli stati-credenza e mostrarono che era possibile orientare un pezzo posto su un tavolo da una posizione iniziale arbitraria me-

diante una sequenza ben progettata di azioni di ribaltamento. Metodi più pratici, basati su una serie di barriere diagonali orientate con precisione su un nastro trasportatore, utilizzano le stesse idee algoritmiche (Wiegley *et al.*, 1996).

L'approccio basato sugli stati-credenza fu reinventato nel contesto dei problemi di ricerca senza sensori e parzialmente osservabili da parte di Genesereth e Nourbakhsh (1993). Ulteriore lavoro fu svolto sui problemi senza sensori nella comunità di pianificazione basata sulla logica (Goldman e Boddy, 1996; Smith e Weld, 1998). Questo lavoro ha messo in evidenza rappresentazioni concise per stati-credenza, come è spiegato nel Capitolo 1. Bonet e Geffner (2000) introdussero le prime euristiche efficaci per ricerche con stato-credenza; queste furono poi raffinate da Bryce *et al.* (2006). L'approccio incrementale alla ricerca con stati-credenza, in cui le soluzioni sono costruite in modo incrementale per sottoinsiemi di stati all'interno di ogni stato-credenza, fu studiato nella letteratura sulla pianificazione da Kurien *et al.* (2002); diversi nuovi algoritmi incrementali sono stati presentati per problemi non deterministici e parzialmente osservabili da Russell e Wolfe (2005). Altri riferimenti per la pianificazione in ambienti stocastici e parzialmente osservabili sono forniti nel Capitolo 17.

Gli algoritmi per l'esplorazione di spazi degli stati sconosciuti sono stati studiati per molti secoli. La ricerca in profondità in un labirinto reversibile può essere implementata non staccando mai la mano sinistra dal muro; per evitare i cicli basta marcare ogni incrocio. Il problema più generale dell'esplorazione dei **grafi di Eulero** (in cui ogni nodo ha un numero uguale di archi entranti e uscenti) è stato risolto da un algoritmo proposto da Hierholzer (1873).

Il primo studio approfondito degli algoritmi per il problema dell'esplorazione di grafi arbitrari fu svolto da Deng e Papadimitriou (1990), che hanno sviluppato un algoritmo del tutto generale, ma hanno anche mostrato che non si può avere un rapporto di competitività limitato nell'esplorazione di grafi generici. Papadimitriou e Yannakakis (1991) hanno esaminato il problema di trovare cammini verso un obiettivo in ambienti di pianificazione geometrici (in cui tutte le azioni sono reversibili) mostrando che nel caso di ostacoli quadrati si può avere un rapporto di competitività piccolo, mentre la presenza di ostacoli rettangolari generici impedisce di ottenere un rapporto limitato (Figura 4.20).

In un ambiente dinamico, lo stato del mondo può cambiare spontaneamente senza che l'agente esegua

alcuna azione. Per esempio, l'agente può pianificare un itinerario di guida ottimo da A a B, ma un incidente o un traffico insolitamente denso può rovinare il piano. Gli algoritmi di ricerca incrementale come Lifelong Planning A* (Koenig *et al.*, 2004) e D* Lite (Koenig e Likhachev, 2002) affrontano questa situazione.

L'algoritmo LRTA* fu sviluppato da Korf (1990) come parte dello studio sulla **ricerca in tempo reale**, rivolta agli ambienti in cui l'agente deve agire dopo aver condotto la ricerca per un tempo limitato (una situazione molto comune nei giochi a due giocatori). LRTA* in effetti è un caso speciale degli algoritmi di apprendimento per rinforzo per ambienti stocastici (Barto *et al.*, 1995). La sua politica di ottimismo in condizioni di incertezza (andare sempre nello stato

più vicino non visitato) può risultare in un'esplorazione che, nel caso non informato, è meno efficiente di una semplice ricerca in profondità (Koenig, 2000). Dasgupta *et al.* (1994) mostrano che la ricerca online ad approfondimento iterativo è ottimamente efficiente quando si deve trovare un obiettivo in un albero uniforme senza informazione euristica.

Sono state sviluppate molte varianti informate di LRTA* con diversi metodi per condurre la ricerca e aggiornare l'informazione nella porzione conosciuta del grafo (Pemberton e Korf, 1992). A tutt'oggi, non c'è una buona comprensione teorica di come trovare obiettivi con efficienza ottimale usando informazione euristica. Sturtevant e Bulitko (2016) forniscono un'analisi di alcuni problemi che si verificano nella pratica.

CAPITOLO

5

- 5.1 La teoria dei giochi
- 5.2 Decisioni ottime nei giochi
- 5.3 Ricerca alfa-beta euristica
- 5.4 Ricerca ad albero
Monte Carlo
- 5.5 Giochi stocastici
- 5.6 Giochi parzialmente
osservabili
- 5.7 Limitazioni degli algoritmi
di ricerca per i giochi
- 5.8 Riepilogo
Note storiche
e bibliografiche

Ricerca con avversari e giochi

In cui esaminiamo gli ambienti in cui agenti pianificano contro di noi.

In questo capitolo esaminiamo gli **ambienti competitivi**, in cui vi sono due o più agenti con obiettivi in conflitto, che danno origine a problemi di **ricerca con avversari** (*adversarial search*). Anziché trattare il caos che si verifica nelle schermaglie del mondo reale, ci concentreremo su giochi, come gli scacchi, il Go e il poker. Per gli studiosi di IA, la natura semplificata di questi giochi è un vantaggio: lo stato di un gioco è facile da rappresentare e gli agenti di solito possono effettuare solo un numero limitato di azioni i cui effetti sono definiti da regole precise. I giochi fisici, come croquet e hockey su ghiaccio, hanno descrizioni più complicate, una gamma più vasta di possibili azioni e regole alquanto imprecise per definirne la legalità. Con l'eccezione del calcio per i robot, i giochi fisici non hanno suscitato molto interesse nella comunità dell'IA.

economia

potatura

funzione di valutazione

informazione imperfetta

informazione perfetta
a somma zeromossa
posizione

5.1 La teoria dei giochi

Esistono almeno tre approcci che possiamo utilizzare negli ambienti multiagente. Il primo approccio, appropriato quando il numero di agenti è molto grande, consiste nel considerarli in aggregato come un'**economia**, il che ci consente di svolgere varie attività, per esempio predire che la crescita della domanda causerà l'aumento dei prezzi, senza dover predire l'azione di un singolo agente.

In secondo luogo, potremmo considerare gli agenti avversari semplicemente come una parte dell'ambiente, un elemento che rende l'ambiente non deterministico. Ma se modelliamo gli avversari in modo simile a come facciamo per eventi privi di controllo, come la pioggia che a volte cade e a volte no, non afferriamo il concetto che i nostri avversari tentano attivamente di sconfiggerci, mentre la pioggia non ha tale intenzione.

Il terzo approccio consiste nel modellare esplicitamente gli agenti avversari con le tecniche di ricerca basate su alberi di gioco. Ed è questo l'argomento del presente capitolo. Iniziamo con una classe ristretta di giochi e definiamo la mossa ottima e un algoritmo per trovarla: la ricerca minimax, una generalizzazione della ricerca AND-OR (Figura 4.11). Mostriamo che con la **potatura** (*pruning*) la ricerca diviene più efficiente perché si ignorano porzioni dell'albero di ricerca che non fanno differenza per la mossa ottima. Per giochi non banali, solitamente non abbiamo abbastanza tempo per essere certi di trovare la mossa ottima (anche con la potatura); dovremo prima o poi interrompere la ricerca.

Per ogni stato in cui scegliamo di interrompere la ricerca, ci chiediamo chi sta vincendo. Per rispondere a questa domanda possiamo scegliere se applicare una **funzione di valutazione** euristica per stimare chi vince in base alle caratteristiche dello stato (Paragrafo 5.3), oppure considerare la media dei risultati di molte simulazioni rapide del gioco a partire dallo stato attuale fino alla fine (Paragrafo 5.4).

Nel Paragrafo 5.5 esaminiamo giochi che includono un elemento legato al caso (per esempio con il lancio di dadi o il rimescolamento delle carte) e nel Paragrafo 5.6 trattiamo i giochi con **informazione imperfetta** (come poker e bridge, in cui non tutte le carte sono visibili a tutti i giocatori).

5.1.1 Giochi a somma zero a due giocatori

I giochi studiati più spesso nel campo dell'IA (come gli scacchi e il Go) sono quelli che gli studiosi di teoria dei giochi chiamano deterministici, a due giocatori, a turni, con **informazione perfetta, a somma zero**. “Informazione perfetta” è un altro modo per dire “completamente osservabile”¹ e “a somma zero” significa che ciò che va a vantaggio di un giocatore danneggia l'altro: non è possibile un risultato “win-win” in cui tutti vincono. Per i giochi utilizziamo spesso il termine **mossa** come sinonimo di azione e **posizione** come sinonimo di stato.

Consideriamo un gioco a due giocatori, che chiameremo MAX e MIN per ragioni che saranno palesi tra poco. MAX muove per primo, dopodiché entrambi giocano a turno fino alla fine della partita: a questo punto vengono assegnati premi al vincitore e penalità al perdente. Un gioco può essere definito formalmente con le seguenti componenti.

- S_0 : lo **stato iniziale**, che specifica come è configurato il gioco in partenza.
- **DEVE-MUOVERE(s)**: il giocatore a cui tocca muovere nello stato s .
- **AZIONI(s)**: l'insieme delle mosse legali nello stato s .

¹ Alcuni autori fanno una distinzione, parlando di “gioco con informazione imperfetta” per i giochi come il poker in cui i giocatori hanno informazioni private sulle loro carte che gli altri giocatori non hanno, e “gioco parzialmente osservabile” per i giochi come StarCraft II in cui ogni giocatore può vedere l'ambiente circostante, ma non quello più lontano.

- **RISULTATO**(s, a): il **modello di transizione**, che definisce lo stato risultante dall'esecuzione dell'azione a nello stato s .
- **È-TERMINALE**(s): un **test di terminazione**, che restituisce vero se la partita è finita e falso altrimenti. Gli stati che fanno finire la partita sono chiamati **stati terminali**.
- **UTILITÀ**(s, p): una **funzione di utilità** (chiamata anche funzione obiettivo o funzione di payoff) che definisce il valore numerico finale per il giocatore p quando il gioco termina nello stato terminale s . Negli scacchi i possibili risultati sono vittoria, sconfitta o pareggio e i valori corrispondenti sono 1, 0 o $1/2$.² Altri giochi hanno una gamma più vasta di possibili risultati; per esempio, nel backgammon il payoff va da 0 a +192.

modello di transizione**test di terminazione**
stati terminali**grafo dello spazio degli stati****albero di ricerca**
albero di gioco

In modo simile a quanto abbiamo visto nel Capitolo 3, lo stato iniziale, la funzione AZIONI e la funzione RISULTATO definiscono il **grafo dello spazio degli stati**, un grafo in cui i vertici sono stati, gli archi sono mosse e uno stato può essere raggiunto da più cammini. Come nel Capitolo 3, su parte di tale grafo possiamo sovrapporre un **albero di ricerca** per determinare quale mossa effettuare. Definiamo **albero di gioco** un albero di ricerca che segue ogni sequenza di mosse fino a raggiungere uno stato terminale. L'albero di gioco potrebbe essere infinito se lo spazio degli stati è illimitato o se le regole del gioco consentono di ripetere le posizioni all'infinito.

La Figura 5.1 mostra una parte dell'albero di gioco per il gioco del tris (o tic-tac-toe). Partendo dallo stato iniziale, MAX ha nove possibili mosse. Il gioco si alterna tra il piazzamento

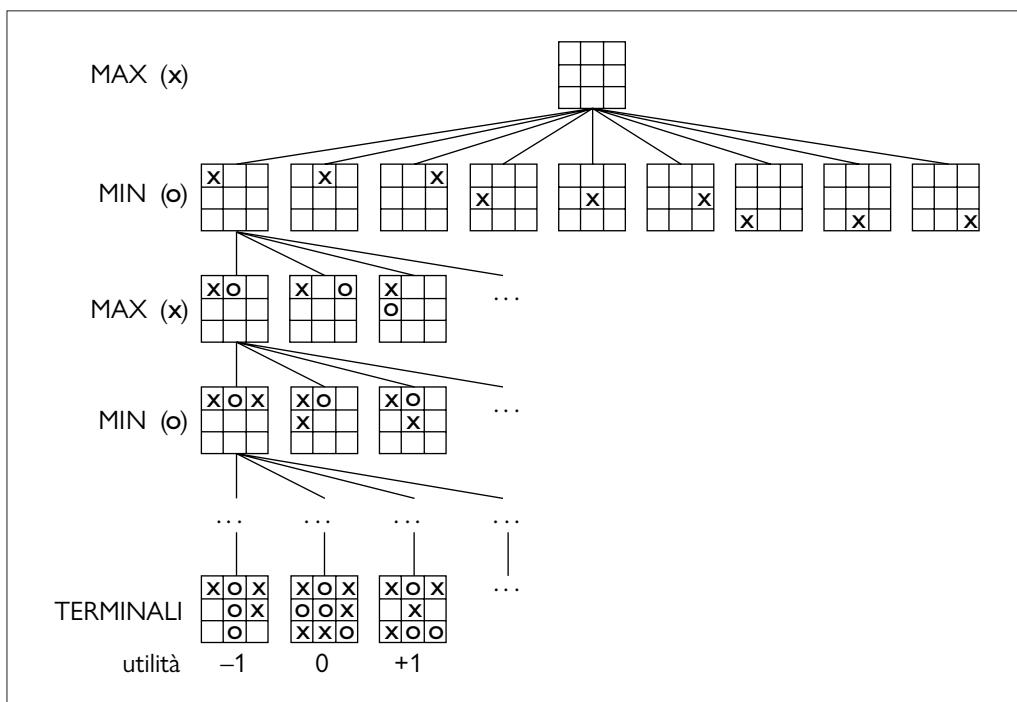


Figura 5.1 Un albero di gioco (parziale) per il gioco del tris o tic-tac-toe. Il nodo più in alto è lo stato iniziale, e MAX muove per primo, ponendo una X in un riquadro vuoto. Abbiamo riportato una parte dell'albero, indicando le mosse alternate di MIN (che usa il simbolo O) e MAX (X), fino al raggiungimento degli stati terminali, a cui possono essere associati valori di utilità in base alle regole del gioco.

² Il gioco degli scacchi è considerato un gioco a somma zero, anche se in ogni partita la somma dei risultati per i due giocatori è +1, non zero. Sarebbe più accurato parlare di "gioco a somma costante", ma tradizionalmente si dice "a somma zero", che è sensato se si immagina che ogni giocatore sia inizialmente penalizzato di un valore $1/2$.

di una X da parte di MAX e di una O da parte di MIN finché non viene raggiunto un nodo foglia che corrisponde a uno stato terminale, in cui un giocatore ha fatto tris oppure sono state riempite tutte le caselle. Il numero associato a ciascun nodo foglia indica il valore di utilità di quello stato terminale dal punto di vista di MAX; i valori alti sono buoni per MAX e cattivi per MIN (ed ecco spiegato come abbiamo assegnato i nomi ai giocatori).

Per il gioco del tris l'albero è relativamente piccolo, meno di $9! = 362.880$ nodi terminali (con soli 5.478 stati distinti), mentre per gli scacchi ci sono oltre 10^{40} nodi, perciò è meglio pensare all'albero di gioco come a un costrutto teorico che non possiamo realizzare nel mondo fisico.

5.2 Decisioni ottime nei giochi

MAX vuole trovare una sequenza di azioni che porti a una vittoria, ma MIN ha qualcosa da obiettare al riguardo. Questo significa che la strategia di MAX deve essere un piano condizionale, cioè una strategia che specifica una reazione a ognuna delle possibili mosse di MIN. Nei giochi in cui il risultato è binario (vittoria o sconfitta), per generare il piano condizionale potremmo usare la ricerca AND-OR (cfr. Paragrafo 4.3.2). In effetti, per questi tipi di giochi la definizione di una strategia vincente è identica alla definizione di una soluzione per un problema di pianificazione non deterministico: in entrambi i casi il risultato desiderabile deve essere garantito a prescindere dal comportamento della controparte. Per i giochi con più risultati possibili, invece, ci serve un algoritmo un po' più generale chiamato **ricerca minimax**.

ricerca minimax

strato

valore minimax

Consideriamo il gioco banale illustrato nella Figura 5.2. Le mosse possibili per MAX nel nodo radice sono etichettate a_1 , a_2 e a_3 . Le possibili risposte di MIN alla mossa a_1 sono b_1 , b_2 , b_3 e così via. Questo particolare gioco termina dopo una sola mossa di MAX e una di MIN (in alcuni giochi si parla di “mossa” quando entrambi i giocatori hanno compiuto un’azione, altrimenti si utilizza il termine **ply** o **strato** per indicare in modo non ambiguo una mossa compiuta da un solo giocatore che ci fa scendere un livello più in basso nell’albero del gioco). Le utilità degli stati terminali in questa partita vanno da 2 a 14.

Dato un albero di gioco, è possibile determinare la strategia ottima calcolando il **valore minimax** di ogni stato, che scriviamo come $\text{MINIMAX}(s)$. Il valore minimax è l’utilità (per MAX) di trovarsi nello stato corrispondente, assumendo che entrambi gli agenti giochino in modo ottimo da lì al termine della partita. Il valore minimax di uno stato terminale è sem-

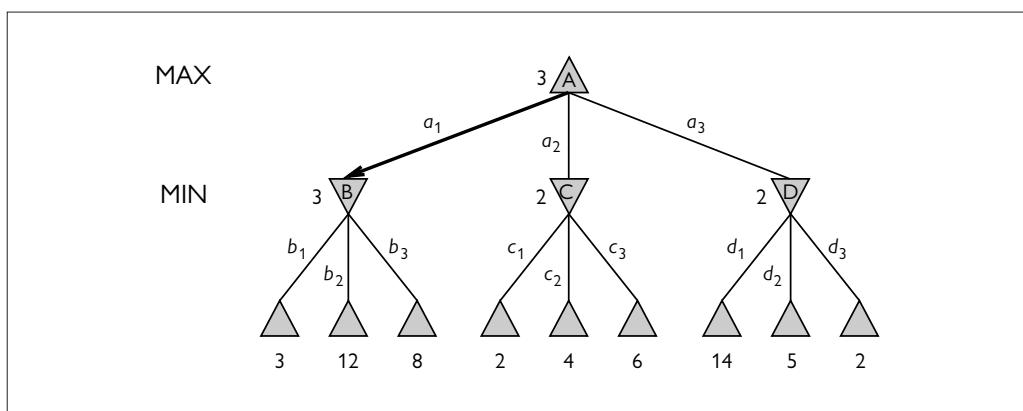


Figura 5.2 Un albero di gioco a due strati. I nodi Δ sono “nodi MAX”, in cui è il turno di MAX a muovere, quelli ∇ sono “nodi MIN”. I nodi terminali indicano i valori di utilità per MAX; gli altri sono etichettati con i loro valori minimax. Alla radice la mossa migliore di MAX è a_1 , perché porta al successore con il più alto valore minimax, mentre la risposta migliore per MIN è b_1 , perché porta al successore con valore minimax più basso.

plicemente la sua utilità. In uno stato non terminale, MAX al suo turno preferirà muoversi verso uno stato di valore massimo, mentre MIN al suo turno preferirà muoversi verso uno stato di valore minimo (cioè verso uno stato con valore minimo per MAX e valore massimo per MIN). Abbiamo quindi:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITÀ}(s, \text{MAX}) & \text{se È-TERMINALE}(s) \\ \text{MAX}_{a \in \text{AZIONI}(s)} \text{MINIMAX}(\text{RISULTATO}(s, a)) & \text{se DEVE-MUOVERE }(s) = \text{MAX} \\ \text{MIN}_{a \in \text{AZIONI}(s)} \text{MINIMAX}(\text{RISULTATO}(s, a)) & \text{se DEVE-MUOVERE }(s) = \text{MIN} \end{cases}$$

Ora applichiamo queste definizioni all'albero di gioco della Figura 5.2. I nodi terminali al livello più basso ottengono i loro valori di utilità dalla funzione UTILITÀ del gioco. Il primo nodo MIN, etichettato *B*, ha tre successori con valori 3, 12 e 8, cosicché il suo valore minimax è 3. Similmente, gli altri due nodi MIN hanno un valore minimax di 2. La radice è un nodo MAX; i suoi successori hanno valori minimax di 3, 2 e 2; quindi la radice ha un valore minimax di 3. Possiamo anche individuare la **decisione minimax** alla radice: l'azione a_1 è la scelta ottima per MAX, perché porta allo stato successore con il più alto valore minimax.

Questa definizione di gioco ottimo per MAX presume che anche MIN giochi in modo ottimo. Ma che cosa succede se MIN non gioca in modo ottimo? In questo caso MAX si comporterà almeno altrettanto bene di come si comporterebbe contro un giocatore ottimo, forse anche meglio. Tuttavia, questo non significa che sia sempre meglio effettuare la mossa ottima minimax quando si affronta un avversario subottimo. Consideriamo una situazione in cui il gioco ottimo di entrambe le parti porterà a un pareggio, ma vi è una mossa rischiosa per MAX che porta a uno stato in cui vi sono 10 possibili mosse di risposta da parte di MIN. Tali mosse appaiono tutte ragionevoli, ma 9 di esse comportano una perdita per MIN e 1 comporta una perdita per MAX. Se MAX ritiene che MIN non abbia potenza di calcolo sufficiente per scoprire la mossa ottima, potrebbe essere disposto a effettuare la mossa rischiosa, ritenendo che 9/10 di possibilità di una vittoria siano meglio di un pareggio certo.

decisione minimax

5.2.1 L'algoritmo di ricerca minimax

Ora che siamo in grado di calcolare $\text{MINIMAX}(s)$, possiamo trasformarlo in un algoritmo di ricerca che trova la mossa migliore per MAX provando tutte le azioni e scegliendo quella per cui lo stato di arrivo ha il valore MINIMAX più alto. L'algoritmo è mostrato nella Figura 5.3: è un algoritmo ricorsivo che percorre l'albero verso il basso fino alle foglie e poi “porta su” (*back up*) i valori minimax nella fase di ritorno della ricorsione. Per esempio, nella Figura 5.2 l'algoritmo prima scende fino ai tre nodi in basso a sinistra e usa su di essi la funzione UTILITÀ per scoprire che i loro valori sono rispettivamente 3, 12 e 8. Poi prende il minimo di questi valori, 3, e lo restituisce come valore “portato su” al nodo *B*. Un processo simile fornisce i valori 2 per *C* e 2 per *D*. Infine, il massimo tra 3, 2 e 2 fornisce il valore 3 della radice.

L'algoritmo minimax esegue una esplorazione completa in profondità dell'albero di gioco. Se la profondità massima dell'albero è m , e in ogni punto ci sono b mosse legali, la complessità temporale dell'algoritmo sarà $O(b^m)$. Quella spaziale è $O(bm)$ se l'algoritmo genera i successori tutti insieme, oppure $O(m)$ se li genera uno per volta (cfr. Paragrafo 3.4.3). Data la complessità esponenziale, l'algoritmo MINIMAX non è utilizzabile per giochi complessi; per esempio, il gioco degli scacchi ha un fattore di ramificazione di circa 35 e la partita media ha una profondità di circa 80 strati, e non è praticabile effettuare ricerche con $35^{80} \approx 10^{123}$ stati. Possiamo però approssimare in vari modi l'analisi minimax per ricavarne algoritmi più pratici.

```

function RICERCA-MINIMAX(gioco, stato) returns un’azione
  giocatore  $\leftarrow$  gioco.DEVE-MUOVERE(stato)
  valore, mossa  $\leftarrow$  VALORE-MAX(gioco, stato)
  return mossa

function VALORE-MAX(gioco, stato) returns una coppia (utilità, mossa)
  if gioco.È-TERMINALE(stato) then return gioco.UTILITÀ(stato, giocatore), null
  v  $\leftarrow$   $-\infty$ 
  for each a in gioco.AZIONI(stato) do
    v2, a2  $\leftarrow$  VALORE-MIN(gioco, gioco.RISULTATO(stato,a))
    if v2  $>$  v then
      v, mossa  $\leftarrow$  v2, a
  return v, mossa

function VALORE-MIN(gioco, stato) returns una coppia (utilità, mossa)
  if gioco.È-TERMINALE(stato) then return gioco.UTILITÀ(stato, giocatore), null
  v  $\leftarrow$   $+\infty$ 
  for each a in gioco.AZIONI(stato) do
    v2, a2  $\leftarrow$  VALORE-MAX(gioco, gioco.RISULTATO(stato,a))
    if v2  $<$  v then
      v, mossa  $\leftarrow$  v2, a
  return v, mossa

```

Figura 5.3 Un algoritmo per calcolare la mossa ottima mediante minimax, cioè la mossa che porta al risultato di massima utilità, sotto l’ipotesi che lo scopo dell’avversario sia minimizzare la stessa funzione di utilità. Le funzioni VALORE-MAX e VALORE-MIN attraversano l’intero albero di gioco fino alle foglie per determinare il valore “portato su” di uno stato e la mossa per raggiungerlo.

5.2.2 Decisioni ottime nei giochi multiplayer

Molti giochi popolari permettono la presenza di più di due giocatori. Vediamo come si può estendere il concetto di minimax ai giochi multiplayer: dal punto di vista tecnico non ci sono problemi, ma questo permette di sollevare nuove questioni interessanti.

Prima di tutto, dobbiamo sostituire il valore singolo di ogni nodo con un *vettore* di valori. Per esempio, in un gioco a tre con i giocatori *A, B e C*, a ogni nodo si dovrà associare un vettore $\langle v_A, v_B, v_C \rangle$. Per gli stati terminali, il vettore fornirà l’utilità dello stato dal punto di vista di ogni giocatore (nei giochi a due giocatori a somma zero, il vettore è superfluo perché i due valori sono sempre uno l’opposto dell’altro). Il modo più semplice di implementare tutto ciò è far sì che la funzione UTILITÀ restituiscia un vettore.

Ora consideriamo gli stati non terminali: prendiamo il nodo marcato con la *X* nell’albero di gioco della Figura 5.4. In quello stato, il giocatore *C* deve decidere cosa fare. Le due scelte portano a stati terminali con vettori di utilità $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ e $\langle v_A = 4, v_B = 2, v_C = 3 \rangle$. Dato che 6 è maggiore di 3, *C* dovrebbe scegliere la prima mossa. Questo significa che, se si raggiunge lo stato *X*, la mossa successiva porterà a uno stato terminale con utilità $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$. Ne consegue che il valore di *X* “portato su” è questo vettore. In generale, il valore che viene passato verso l’alto a un nodo *n* è il vettore di utilità dello stato successore più favorevole a chi sta scegliendo la mossa in *n*.

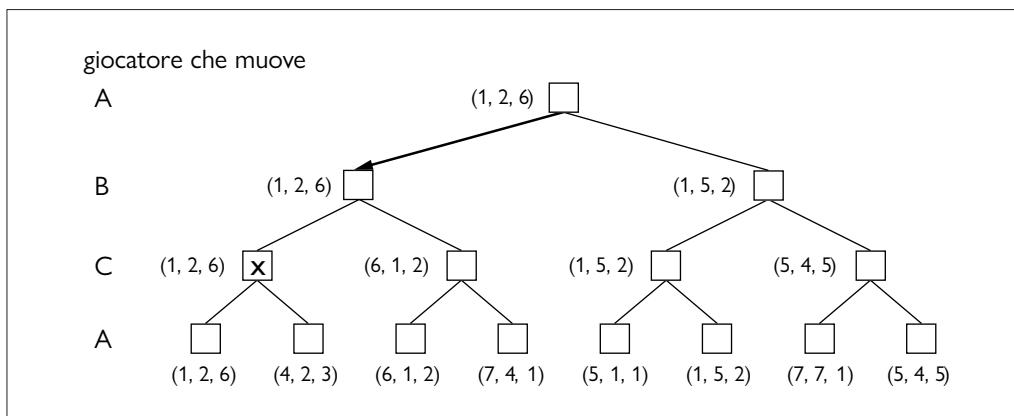


Figura 5.4
I primi tre strati (ply) di un albero di gioco con tre giocatori (A, B, C). Ogni nodo è etichettato con i valori dal punto di vista di ogni giocatore. La mossa migliore partendo dalla radice è evidenziata con la freccia.

Chiunque abbia provato un gioco multiplayer, come Diplomacy o I coloni di Catan, si è ben presto accorto che le cose sono molto più complicate del caso a due giocatori. I giochi multiplayer normalmente prevedono **alleanze** tra i giocatori, esplicitamente formalizzate o no, che vengono strette e poi spezzate durante lo svolgimento della partita. Come possiamo considerare un comportamento simile? Le alleanze sono una conseguenza naturale delle strategie ottime di ogni giocatore? I risultati dicono che potrebbe essere così. Supponiamo per esempio che A e B si trovino in una posizione debole e C in una forte. In questo caso la scelta ottima per A e B spesso consiste nell'attaccare C invece che attaccarsi a vicenda, per impedirgli di distruggere entrambi individualmente. In questo modo, la collaborazione emerge da un comportamento puramente egoistico. Naturalmente, non appena C si indebolisce sotto l'attacco congiunto, l'alleanza perde valore, e sia A che B potrebbero violare gli accordi.

In alcuni casi, la presenza di alleanze esplicite rende semplicemente più concreto ciò che si sarebbe verificato comunque. In altri casi la rottura di un'alleanza costituisce una macchia sulla propria reputazione, per cui i giocatori devono bilanciare i vantaggi immediati del tradimento con l'inconveniente a lungo termine di essere notoriamente inaffidabili. Nel Paragrafo 18.2 verranno trattati ancora questi aspetti.

Se il gioco non è a somma zero, si può verificare una collaborazione anche con due soli giocatori. Supponiamo per esempio che ci sia uno stato terminale con guadagni $\langle v_A = 1000, v_B = 1000 \rangle$, e che 1000 sia il valore di utilità più alto possibile per ogni giocatore. Allora la strategia ottima per ambedue i giocatori è raggiungere questo stato; questo significa che i due cooperrano automaticamente per raggiungere un obiettivo desiderabile per entrambi.

5.2.3 Potatura alfa-beta

Il numero di stati del gioco cresce esponenzialmente con la profondità dell'albero. Nessun algoritmo può eliminare l'esponente, ma a volte è possibile dimezzarlo calcolando la decisione minimax corretta senza esaminare ogni stato, attraverso la **potatura** (cfr. Paragrafo 3.5.3) di grandi porzioni dell'albero che non influenzano il risultato finale. Esamineremo una tecnica particolare chiamata **potatura alfa-beta**.

Consideriamo ancora l'albero a due strati della Figura 5.2. Calcoliamo ancora una volta la decisione ottima, ponendo particolare attenzione alle informazioni che abbiamo a disposizione in ogni punto del processo. I vari passi sono illustrati nella Figura 5.5. Il risultato è che possiamo arrivare alla decisione minimax senza valutare due dei nodi foglia.

Un altro modo di considerare questa tecnica è come una semplificazione della formula per calcolare MINIMAX. Siano x e y i valori dei due successori non valutati del nodo C nella Figura 5.5. Il valore del nodo radice è dato da:

alleanza

potatura alfa-beta

$$\begin{aligned}
 \text{MINIMAX}(radice) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\
 &= \max(3, \min(2, x, y), 2) \\
 &= \max(3, z, 2) \quad \text{dove } z = \min(2, x, y) \leq 2 \\
 &= 3.
 \end{aligned}$$

In altre parole, il valore della radice e di conseguenza la decisione minimax sono *indipendenti* dai valori delle foglie x e y , che quindi possono essere potate.

La potatura alfa-beta può essere applicata ad alberi di qualunque profondità, e spesso invece di foglie è possibile potare interi sottoalberi. Il principio generale è questo: considerate un nodo n da qualche parte nell'albero (Figura 5.6), tale che Giocatore abbia la facoltà di raggiungerlo. Se Giocatore ha una scelta migliore allo stesso livello (per esempio m' nella Figura 5.6) o in qualsiasi punto di livello più alto nell'albero (per esempio m nella Figura 5.6), allora non raggiungerà mai n . Possiamo quindi potare n non appena abbiamo raccolto

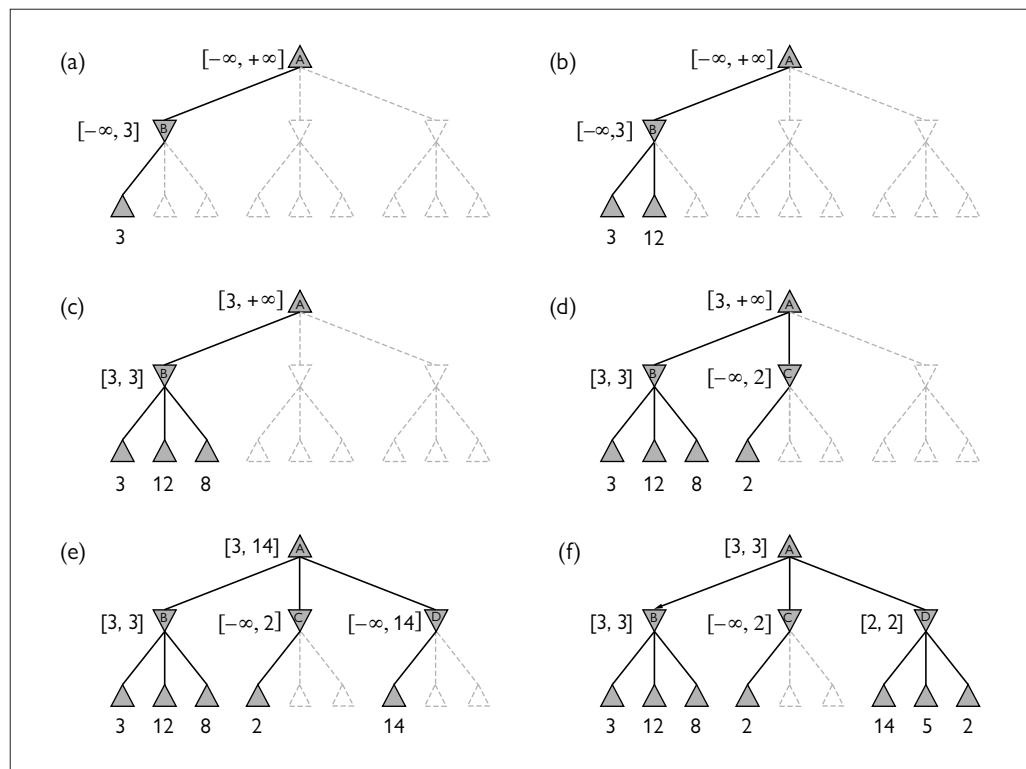


Figura 5.5 Fasi del calcolo della decisione ottima per l'albero di gioco della Figura 5.2: in ogni punto viene indicato l'intervallo dei possibili valori di ogni nodo. (a) La prima foglia sotto B ha valore 3; ne consegue che B , che è un nodo MIN, ha un valore *al più* di 3. (b) La seconda foglia sotto B ha valore 12; MIN la eviterebbe, per cui il valore di B è ancora *al più* 3. (c) La terza foglia sotto B ha valore 8; ora abbiamo considerato tutti gli stati successori di B , per cui possiamo dire che il valore di B è esattamente 3. Ne possiamo dedurre che il valore della radice è *almeno* 3, perché MAX ha appunto a disposizione tale scelta. (d) La prima foglia sotto C ha valore 2; ne consegue che C , che è un nodo MIN, ha un valore *al più* di 2. Ma sappiamo che C vale 3, per cui MAX non sceglierrebbe C in nessun caso: non c'è quindi alcuna ragione di considerare gli altri stati successori di C . Questo è un esempio di potatura alfa-beta. (e) La prima foglia sotto D ha valore 14, cosicché D vale *al più* 14: questo valore è ancora superiore alla migliore alternativa per MAX (che è 3), per cui dobbiamo continuare a esplorare gli stati successori di D . Notate che ora conosciamo gli estremi di tutti i successori della radice, e possiamo dire che il suo valore massimo può essere 14. (f) Il secondo successore di D vale 5, quindi dobbiamo ancora continuare a esplorare. Il terzo successore di D vale 2, per cui D vale esattamente 2. La decisione di MAX alla radice è muovere nello stato B , che offre un valore di 3.

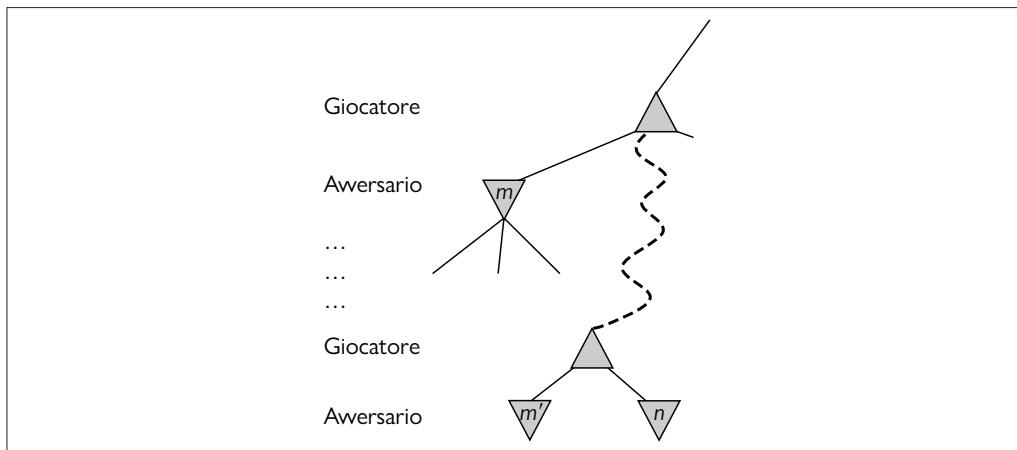


Figura 5.6
Potatura alfa-beta:
il caso generale.
Se per Giocatore
 m o m' è
preferibile a n ,
durante il gioco
non arriveremo
mai a n .

abbastanza informazioni (esaminando alcuni dei suoi discendenti) da raggiungere questa conclusione.

Ricordate che quella minimax è una ricerca in profondità, per cui in ogni momento dobbiamo considerare solo i nodi lungo un singolo cammino dell'albero. La potatura alfa-beta prende il suo nome dai due parametri addizionali in VALORE-MAX(*giocostato*, α, β) (Figura 5.7) che descrivono i limiti sui valori “portati su” in un qualsiasi punto del cammino:

α = il valore della scelta migliore (quella con valore più alto) per MAX che abbiamo trovato sin qui in un qualsiasi punto di scelta lungo il cammino. Possiamo considerare: α = “almeno”.

β = il valore della scelta migliore (quella con valore più basso) per MIN che abbiamo trovato sin qui in un qualsiasi punto di scelta lungo il cammino. Possiamo considerare: β = “al più”.

La ricerca alfa-beta aggiorna i valori di α e β a mano a mano che procede e pota i rami restanti che escono da un nodo (ovvero, fa terminare le chiamate ricorsive) non appena determina che il valore del nodo è peggiore di quello di α per MAX o, rispettivamente, di β per MIN. L’algoritmo completo è riportato nella Figura 5.7. Nella Figura 5.5 viene tracciata l’esecuzione dell’algoritmo su un albero di gioco.

5.2.4 Ordinamento delle mosse

L’efficacia della potatura alfa-beta dipende fortemente dall’ordine in cui sono esaminati gli stati. Per esempio, nella Figura 5.5(e) e (f), non abbiamo potuto potare alcun successore di D perché tutti i successori peggiori (dal punto di vista di MIN) sono stati generati prima. Se il terzo successore di D fosse stato generato per primo, con valore 2, saremmo stati in grado di potare gli altri due successori. Questo ci suggerisce che sarebbe una buona idea cercare di esaminare per primi i successori migliori. Se fosse possibile fare ciò in modo perfetto, la ricerca alfa-beta per scegliere la mossa migliore dovrebbe esaminare solo $O(b^{m/2})$ nodi, invece degli $O(b^m)$ richiesti dalla ricerca minimax. Questo significa che il fattore di ramificazione effettivo diventa \sqrt{b} invece di b : nel caso degli scacchi, 6 invece di 35.

In altre parole, si può dire che la ricerca alfa-beta con ordinamento perfetto delle mosse può risolvere un albero profondo circa il doppio di quello che può risolvere minimax nello stesso lasso di tempo. Se i successori sono esaminati in ordine casuale, il numero totale di nodi esaminati sarà circa $O(b^{3m/4})$ per b di grandezza moderata. Naturalmente non è possibile ottenere un ordinamento *perfetto* delle mosse (altrimenti si potrebbe usare la funzione di ordinamento per giocare una partita perfetta!), ma spesso ci si può andare abbastanza vi-

```

function RICERCA-ALFA-BETA(gioco, stato) returns un’azione
  giocatore  $\leftarrow$  gioco.DEVE-MUOVERE(stato)
  valore, mossa  $\leftarrow$  VALORE-MAX(gioco, stato,  $-\infty, +\infty$ )
  return move

function VALORE-MAX(gioco, stato,  $\alpha, \beta$ ) returns una coppia (utilità, mossa)
  if gioco.È-TERMINALE(stato) then return gioco.UTILITÀ(stato, giocatore), null
  v  $\leftarrow -\infty$ 
  for each a in gioco.AZIONI(stato) do
    v2, a2  $\leftarrow$  VALORE-MIN(gioco, gioco.RISULTATO(stato,a),  $\alpha, \beta$ )
    if v2 > v then
      v, mossa  $\leftarrow$  v2, a
       $\alpha$   $\leftarrow$  MAX( $\alpha$ , v)
    if v ≥ β then return v, mossa
  return v, mossa

function VALORE-MIN(gioco, stato,  $\alpha, \beta$ ) returns una coppia (utilità, mossa)
  if gioco.È-TERMINALE(stato) then return gioco.UTILITÀ(stato, giocatore), null
  v  $\leftarrow +\infty$ 
  for each a in gioco.AZIONI(stato) do
    v2, a2  $\leftarrow$  VALORE-MAX(gioco, gioco.RISULTATO(stato,a),  $\alpha, \beta$ )
    if v2 < v then
      v, mossa  $\leftarrow$  v2, a
       $\beta$   $\leftarrow$  MIN( $\beta$ , v)
    if v ≤ α then return v, mossa
  return v, mossa

```

Figura 5.7 L’algoritmo di ricerca alfa-beta. Notate che le funzioni sono le stesse della RICERCA-MINIMAX della Figura 5.3, a parte il fatto che manteniamo dei limiti nelle variabili α e β e li usiamo per tagliare la ricerca quando un valore è al di fuori dei limiti.

cino. Per gli scacchi, una funzione di ordinamento abbastanza semplice (come provare prima le mosse per catturare pezzi, poi quelle per minacciarli, poi quelle in avanti e infine quelle all’indietro) consente di arrivare a un fattore 2 di distanza dal risultato del caso migliore, $O(b^{m/2})$.

Aggiungere schemi dinamici di ordinamento delle mosse, come provare per prime le mosse che si sono rivelate le migliori nel passato, ci porta molto vicini al limite teorico. Il passato potrebbe essere la mossa precedente (spesso le minacce rimangono tali) o potrebbe derivare dalla precedente esplorazione della mossa corrente attraverso un processo di **approfondimento iterativo** (cfr. Paragrafo 3.4.4). Innanzitutto si cerca un solo strato in profondità e si registra l’ordinamento delle mosse in base alle rispettive valutazioni. Poi si cerca uno strato ancora più in profondità, basandosi sull’ordinamento precedente delle mosse; e così via. L’aumento del tempo di ricerca dovuto all’approfondimento iterativo può essere recuperato con gli interessi grazie al miglior ordinamento delle mosse. Le mosse migliori sono note come **mossa killer** e si parla di euristica della mossa killer quando si provano queste mosse per prime.

Nel Paragrafo 3.3 abbiamo notato che cammini ridondanti verso stati ripetuti possono causare un aumento esponenziale del costo di ricerca, e che mantenere una tabella degli stati precedentemente raggiunti può consentire di risolvere questo problema. Nelle ricerche su alberi di gioco, gli stati ripetuti possono verificarsi a causa delle **trasposizioni**, permutazioni diverse della stessa sequenza di mosse che portano tutte nella stessa posizione, e il problema si può risolvere utilizzando una **tabella delle trasposizioni** che memorizza il valore euristico degli stati.

Per esempio, supponiamo che Bianco abbia una mossa b_1 a cui Nero risponderà con n_1 e, dall'altra parte della scacchiera, una mossa indipendente b_2 a cui Nero può contrapporre la risposta n_2 , e di cercare la sequenza di mosse $[b_1, n_1, b_2, n_2]$. Chiamiamo s lo stato risultante. Dopo aver esplorato una grande porzione dell'albero sotto s , troviamo il suo valore “portato su”, che memorizziamo nella tabella delle trasposizioni. Quando poi cerchiamo la sequenza di mosse $[b_2, n_2, b_1, n_1]$, andiamo ancora a finire in s e possiamo cercarne il valore nella tabella anziché ripetere la ricerca. Negli scacchi, l'uso di tabelle di trasposizione è molto efficace, infatti ci consente di raddoppiare la profondità di ricerca a parità di tempo.

Anche con la potatura alfa-beta e un attento ordinamento delle mosse, l'algoritmo minimax non può funzionare per giochi come gli scacchi e Go, perché rimangono comunque troppi gli stati da esaminare nel tempo disponibile. Nel primissimo articolo dedicato al gioco degli scacchi al computer, *Programming a Computer for Playing Chess* (Shannon, 1950), Claude Shannon si rese conto di questo problema e propose due strategie per affrontarlo: una **strategia di Tipo A** considera tutte le mosse possibili fino a una certa profondità nell'albero di ricerca e poi utilizza una funzione di valutazione euristica per stimare l'utilità degli stati a tale profondità. Esamina così una porzione *ampia ma superficiale* dell'albero. Una **strategia di Tipo B** ignora le mosse che non sembrano buone e segue le linee più promettenti “più in profondità possibile”. Esamina così una porzione *profonda ma stretta* dell'albero.

Storicamente, la maggior parte dei programmi per gli scacchi è del Tipo A (che tratteremo nel paragrafo seguente), mentre i programmi per il Go sono più spesso del Tipo B (trattato nel Paragrafo 5.4), perché il fattore di ramificazione è molto più alto nel gioco del Go. Più recentemente, i programmi del Tipo B hanno raggiunto livelli da campioni del mondo in una varietà di giochi, tra cui gli scacchi (Silver *et al.*, 2018).

5.3 Ricerca alfa-beta euristica

Per sfruttare al meglio il nostro tempo di calcolo, che è limitato, possiamo interrompere la ricerca in anticipo e applicare una **funzione di valutazione** euristica agli stati, considerando i nodi non terminali come se lo fossero. In altre parole, sostituiamo la funzione UTILITÀ con EVAL, che stima l'utilità di uno stato. Inoltre sostituiamo il test di terminazione con un **test di taglio** (*cutoff test*) che deve restituire vero per gli stati terminali, ma per il resto è libero di decidere quando interrompere la ricerca in base alla profondità della stessa e a qualsiasi proprietà dello stato che sceglie di considerare. Otteniamo così la formula H-MINIMAX(s, d) per il valore minimax euristico dello stato s alla profondità di ricerca d :

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s, \text{MAX}) & \text{se TEST-TAGLIO}(s, d) \\ \text{MAX}_{a \in \text{AZIONI}(s)} \text{H-MINIMAX}(\text{RISULTATO}(s, a), d+1) & \text{se DEVE-MUOVERE}(s) = \text{MAX} \\ \text{MIN}_{a \in \text{AZIONI}(s)} \text{H-MINIMAX}(\text{RISULTATO}(s, a), d+1) & \text{se DEVE-MUOVERE}(s) = \text{MIN}. \end{cases}$$

trasposizione

tabella delle trasposizioni

strategia di Tipo A

strategia di Tipo B

test di taglio

5.3.1 Funzioni di valutazione

Una funzione di valutazione euristica $\text{EVAL}(s, p)$ restituisce una *stima* dell'utilità attesa dello stato s per il giocatore p , esattamente come le funzioni euristiche del Capitolo 3 restituiscono stime della distanza dall'obiettivo. Per gli stati terminali deve valere l'uguaglianza $\text{EVAL}(s, p) = \text{UTILITÀ}(s, p)$ e per gli stati non terminali la valutazione deve risultare compresa tra una sconfitta e una vittoria: $\text{UTILITÀ}(\text{sconfitta}, p) \leq \text{EVAL}(s, p) \leq \text{UTILITÀ}(\text{vittoria}, p)$.

Per ottenere una buona funzione di valutazione servono anche altri requisiti. In primo luogo, il calcolo non deve richiedere troppo tempo (lo scopo è di velocizzare la ricerca); in secondo luogo, la funzione di valutazione dev'essere fortemente correlata con le effettive probabilità di vincere. Potrebbe sembrare strano parlare di "possibilità di vincere" dato che gli scacchi non sono un gioco basato sul caso: conosciamo lo stato corrente con certezza e non ci sono elementi casuali: se nessuno dei due giocatori commette un errore, il risultato è predeterminato. Ma se la ricerca dev'essere interrotta in stati non terminali, l'algoritmo sarà necessariamente *incerto* sui risultati finali di quegli stati (anche se l'incertezza si potrebbe sciogliere disponendo di risorse di calcolo infinite).

caratteristica

Rendiamo più concreto questo concetto. La maggior parte delle funzioni di valutazione considera varie **caratteristiche** (*feature*) di uno stato – per esempio, negli scacchi avremmo come caratteristiche il numero di pedoni bianchi, il numero di pedoni neri, di regine bianche, di regine nere e così via. Le caratteristiche, prese insieme, definiscono *categorie* o *classi di equivalenza* sugli stati: gli stati di una categoria hanno lo stesso valore per tutte le caratteristiche. Per esempio, una categoria potrebbe contenere tutti i finali di partita con due pedoni contro un pedone. In generale, ogni categoria conterrà stati che portano (con una partita perfetta) alla vittoria, altri che portano al pareggio e altri ancora alla sconfitta. La funzione di valutazione non conosce quali sono questi stati, ma può restituire un singolo valore che stima la *proporzione* di stati che portano a ciascun risultato. Supponiamo per esempio che la nostra esperienza suggerisca che l'82% degli stati nella categoria di due pedoni contro un pedone conducano a una vittoria (utilità +1); il 2% a una sconfitta (0) e il 16% a un pareggio (1/2). In questo caso una valutazione ragionevole per gli stati appartenenti a tale categoria è il **valore atteso**: $(0,82 \times +1) + (0,02 \times 0) + (0,16 \times 1/2) = 0,90$. In via di principio, si può determinare il valore atteso per ogni categoria di stati, ottenendo così una funzione di valutazione applicabile a ogni stato.

valore atteso

Nella pratica, questo tipo di analisi richiede troppe categorie e quindi una quantità eccessiva di esperienza per stimare tutte le probabilità. La maggior parte delle funzioni di valutazione calcola valori separati per ogni caratteristica, *combinandoli* poi insieme per formare il valore finale. Per secoli i giocatori di scacchi hanno elaborato modi per giudicare il valore di una posizione utilizzando soltanto questo concetto. Per esempio, i manuali di scacchi per principianti forniscono un **valore del materiale** approssimativo per ogni pezzo: ogni pedone vale 1, un cavallo o alfiere 3, una torre 5 e la regina 9. Altre caratteristiche come "una buona disposizione dei pedoni" o "la difesa del re" potrebbero valere, poniamo, mezzo pedone. Tutte queste caratteristiche sono semplicemente sommate per ottenere la valutazione di una posizione. Matematicamente questo tipo di valutazione è chiamata **funzione lineare pesata**, perché può essere espressa come:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s),$$

valore del materiale

funzione lineare pesata

dove ogni f_i è una caratteristica (*feature*) della posizione (come il "numero di alfieri bianchi") e ogni w_i è un peso (*weight*) che indica l'importanza di tale caratteristica. I pesi dovrebbero essere normalizzati in modo che la somma pesata sia sempre compresa nell'intervallo fra una sconfitta (0) a una vittoria (1). Un vantaggio sicuro equivalente a un pedone rappresenta una buona possibilità di vittoria, e un vantaggio sicuro equivalente a tre pedoni dovrebbe garantire quasi certamente la vittoria, come si vede nella Figura 5.8(a). Abbiamo detto che

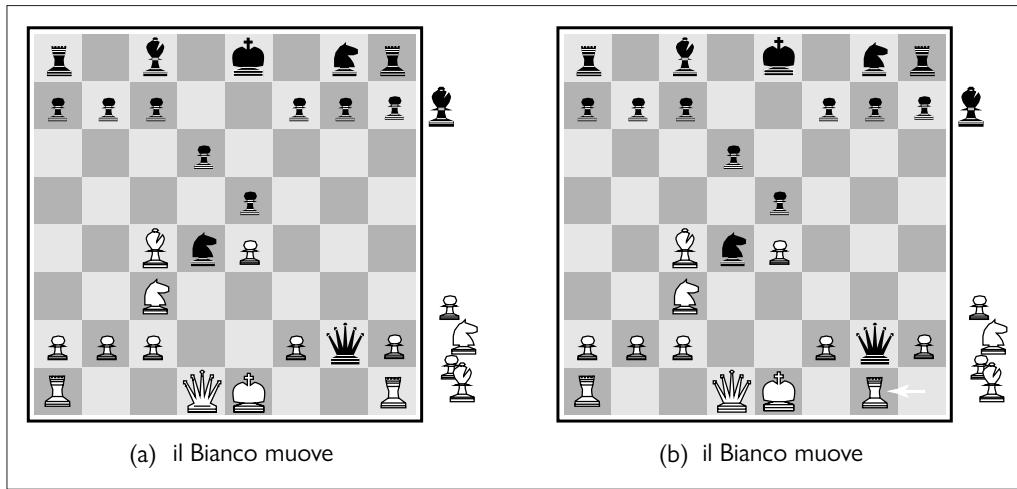


Figura 5.8 Due posizioni sulla scacchiera che differiscono soltanto per la posizione della torre in basso a destra. In (a), il Nero ha un vantaggio di un cavallo e due pedoni, che dovrebbe essere sufficiente per vincere la partita. In (b), il Bianco catturerà la regina, ottenendo un vantaggio che dovrebbe essere sufficiente per vincere.

la funzione di valutazione dovrebbe essere strettamente correlata alle effettive possibilità di vincere, ma non è necessario che sia linearmente correlata: se lo stato s ha il doppio di possibilità di vittoria rispetto allo stato s' , non richiediamo che $\text{EVAL}(s)$ sia il doppio di $\text{EVAL}(s')$, ma soltanto che $\text{EVAL}(s) > \text{EVAL}(s')$.

Sommare i valori delle caratteristiche sembra ragionevole, ma in effetti presume che si accetti l'ipotesi che il contributo di ogni caratteristica sia *indipendente* dal valore delle altre. Per questa ragione, i programmi più recenti per gli scacchi e altri giochi utilizzano combinazioni *non lineari* di caratteristiche: per esempio, una coppia di alfiere potrebbe valere più del doppio di un alfiere singolo, che a sua volta avrà più valore nel finale rispetto a prima – quando la caratteristica *numero di mosse effettuate* è alta, o la caratteristica *numero di pezzi rimanenti* è bassa.

A questo punto potreste chiedervi da dove provengano le caratteristiche e i pesi, che in realtà non fanno parte delle regole degli scacchi, ma rientrano nella cultura del gioco degli scacchi tra esseri umani. Nei giochi in cui non è disponibile questo tipo di conoscenza, i pesi della funzione di valutazione possono essere stimati usando tecniche di apprendimento automatico (cfr. il Capitolo 22 del Volume 2): la loro applicazione agli scacchi ha confermato che un alfiere vale effettivamente circa tre pedoni, e pare che secoli di esperienza umana possano essere replicati in poche ore di apprendimento automatico.

5.3.2 Tagliare la ricerca

Il prossimo passo è modificare RICERCA-ALFA-BETA in modo che invochi la funzione euristica EVAL quando è il momento di tagliare la ricerca. Sostituiamo le due righe della Figura 5.7 che eseguono È-TERMINALE con la seguente:

```
if gioco.TEST-TAGLIO(stato, profondità) then return gioco.EVAL(stato)
```

È necessario inoltre modificare il codice in modo che la *profondità* corrente sia incrementata a ogni chiamata ricorsiva. Il modo più semplice di controllare la quantità di ricerca effettuata è stabilire un limite alla profondità, in modo tale che TEST-TAGLIO($stato, profondità$) restituisca *true* ogni volta che questa supera un valore prefissato d (naturalmente deve anche restituire *true* per tutti gli stati terminali, come faceva È-TERMINALE). La profondità d dev'essere scelta in modo che la scelta di una mossa avvenga nel tempo allocato. Un approccio più robusto è rappresentato dalla ricerca ad approfondimento iterativo (cfr. Capitolo 3). Quan-

do scade il tempo, il programma restituisce la mossa calcolata con la più profonda ricerca completata. In più, se a ogni iterazione di approfondimento registriamo i valori nella tabella delle trasposizioni, le iterazioni successive saranno più veloci e potremo usare la valutazione per migliorare l'ordinamento delle mosse.

Questi semplici approcci possono portare a commettere errori a causa della natura approssimata della funzione di valutazione. Consideriamo ancora la semplice funzione di valutazione per gli scacchi basata sul vantaggio di materiale. Supponiamo che il programma esegua la ricerca fino al limite di profondità, raggiungendo la posizione della Figura 5.8(b), in cui il Nero ha un vantaggio di un cavallo e due pedoni. Il programma assocerebbe quindi a questo stato un valore euristico associato a questo vantaggio, ritenendo che la situazione porterà a una probabile vittoria del Nero. Ma la mossa successiva del Bianco cattura la regina nera senza alcuna compensazione: la posizione quindi è favorevole al Bianco, ma questo si può appurare soltanto guardando avanti.

quiescente

La funzione di valutazione dovrebbe essere applicata solo a posizioni **quiescenti**, quelle in cui non vi sono mosse pendenti (come la cattura della regina) che causerebbero grandi variazioni di valore nella valutazione. Per le posizioni non quiescenti la funzione TEST-TAGLIO restituisce *false* e la ricerca continua fino a raggiungere posizioni quiescenti. Questa ricerca aggiuntiva prende il nome di **ricerca di quiescenza** e talvolta viene ristretta affinché consideri solo certi tipi di mosse, come le catture, che possono risolvere velocemente le situazioni di incertezza.

ricerca di quiescenza

L'**effetto orizzonte** è più difficile da eliminare. Questo problema sorge quando il programma deve considerare una mossa dell'avversario che causa grave danno e non è evitabile, ma può essere rimandato temporaneamente usando tattiche dilatorie. Considerate la posizione mostrata nella Figura 5.9. È chiaro che l'alfiere nero non può scappare. Per esempio, la torre bianca può catturarlo muovendo in h1, poi a1, poi a2; una cattura a profondità di 6 strati. Tuttavia, il Nero dispone di una sequenza di mosse che spinge la cattura dell'alfiere “oltre l'orizzonte”. Supponiamo che il Nero cerchi fino alla profondità di 8 strati. La maggior parte delle mosse del Nero porterà alla cattura dell'alfiere, e perciò saranno contrassegnate come

effetto orizzonte

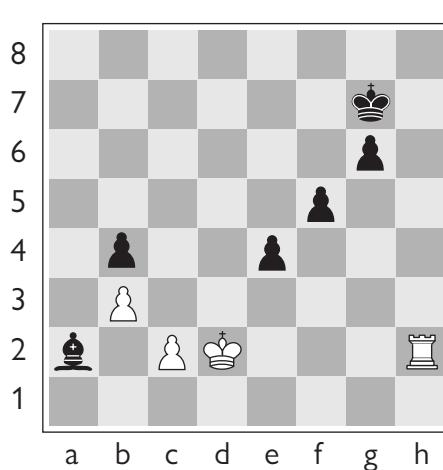


Figura 5.9 L'effetto orizzonte: il Nero deve muovere e il suo alfiere è certamente condannato. Ma il Nero può rimandare l'evento mettendo sotto scacco il re bianco con i suoi pedoni, costringendo il re a catturarli. Questo spinge l'inevitabile perdita dell'alfiere “oltre l'orizzonte”. L'algoritmo di ricerca vede i sacrifici dei pedoni come buone mosse quando in realtà non lo sono.

“cattive” mosse. Tuttavia, il Nero prenderà in considerazione anche la sequenza di mosse che inizia mettendo sotto scacco il re con un pedone, in modo da invitare il re a catturare il pedone. Il Nero può quindi fare la stessa cosa con un altro pedone. Tutto questo richiede un tale numero di mosse che la cattura dell’alfiere non verrà scoperta durante la parte rimanente della ricerca svolta dal Nero. Il Nero pensa che la linea di gioco abbia salvato l’alfiere al prezzo di due pedoni, quando in realtà non ha fatto altro che sprecare due pedoni e rimandare l’inevitabile cattura dell’alfiere al di là dell’orizzonte che è in grado di vedere.

Una strategia per mitigare l’effetto orizzonte è di consentire le **estensioni singole**, mosse che sono “chiaramente migliori” di tutte le altre in una data posizione, anche quando la ricerca sarebbe normalmente interrotta in quel punto. Nel nostro esempio, una ricerca rivelerà che tre mosse della torre bianca – da h2 in h1, poi da h1 ad a1, e poi da a1 alla cattura dell’alfiere in a2 – sono “chiaramente migliori” una per una, perciò anche se una sequenza di mosse di pedone sposta in avanti l’orizzonte, queste mosse chiaramente migliori avranno l’opportunità di estendere la ricerca. In questo modo l’albero diventa più profondo, ma poiché di solito le estensioni singole sono poche, la strategia non aggiunge molti nodi in totale all’albero, e si è dimostrata efficace nella pratica.

estensione singola

5.3.3 Potatura in avanti

La potatura alfa-beta pota i rami dell’albero che non possono influenzare la valutazione finale, mentre la **potatura in avanti** taglia quelle che appaiono cattive mosse, ma potrebbero invece essere buone. La strategia, quindi, fa risparmiare tempo di calcolo ma introduce il rischio di commettere un errore. Nella terminologia di Shannon questa è una strategia di Tipo B. Chiaramente la maggior parte degli esseri umani gioca a scacchi con una strategia di questo tipo, considerando soltanto poche mosse a partire dalla posizione attuale (almeno a livello cosciente).

potatura in avanti

Un approccio alla potatura in avanti è la **ricerca beam** (cfr. Paragrafo 4.1.3): a ogni strato si considera soltanto un “fascio” (beam) delle n migliori mosse (secondo la funzione di valutazione), invece di considerare tutte le mosse possibili. Sfortunatamente questo approccio è piuttosto pericoloso, perché non c’è alcuna garanzia di non potare la mossa migliore.

L’algoritmo PROBCUT, o *probabilistic cut*, “taglio probabilistico” (Buro, 1995) è una versione con potatura in avanti dell’algoritmo di ricerca alfa-beta, che utilizza statistiche ricavate dalla precedente esperienza per ridurre la possibilità che la mossa migliore venga potata. La ricerca alfa-beta pota qualsiasi nodo di cui si possa *dimostrare* che è al di fuori della finestra corrente (α, β). PROBCUT pota anche i nodi che sono *probabilmente* al di fuori della finestra. L’algoritmo determina questa probabilità eseguendo una ricerca poco profonda per calcolare il valore v “portato su” di un nodo e poi utilizzando l’esperienza precedente per stimare la probabilità che un valore di v a profondità d nell’albero sia al di fuori di (α, β) . Buro applicò questa tecnica al suo programma per giocare a Othello, LOGISTELLO, e trovò che una versione di tale programma con PROBCUT batte la versione normale per il 64% delle volte, anche quando la versione normale riceve una disponibilità di tempo doppia.

Un’altra tecnica, la **riduzione delle ultime mosse** (*late move reduction*), funziona sulla base dell’ipotesi che l’ordinamento delle mosse sia stato effettuato bene e quindi che le mosse riportate alla fine nell’elenco delle mosse possibili siano quelle con minori possibilità di essere buone mosse. Anziché potarle completamente, tuttavia, ci limitiamo a ridurre la profondità di ricerca per queste mosse, in modo da risparmiare tempo. Se la ricerca ridotta restituisce un valore superiore al valore α corrente, possiamo rieseguirla a profondità intera.

riduzione delle ultime mosse

Un programma che sfrutta tutte le tecniche che abbiamo descritto sarà in grado di giocare decentemente a scacchi (o ad altri giochi). Supponiamo di aver implementato una funzione di valutazione per gli scacchi, un test di taglio ragionevole con ricerca di quiescenza. Supponiamo anche, dopo mesi di faticosa programmazione, di poter generare e valutare circa un milione

di nodi al secondo su un moderno PC. Il fattore di ramificazione degli scacchi in media è intorno a 35, e 35^5 corrisponde a circa 50 milioni, quindi usando ricerca minimax potremmo guardare in avanti solo di cinque livelli o strati dell'albero di gioco in circa un minuto di calcolo; le regole della competizione non ci darebbero il tempo sufficiente per cercare in sei strati. Per quanto non sia totalmente incompetente, un programma siffatto potrebbe essere facilmente battuto da un giocatore umano medio, che occasionalmente può pianificare sei/otto livelli avanti.

Con la ricerca alfa-beta e una grande tabella delle trasposizioni possiamo arrivare a circa 14 livelli, che corrisponde a un livello di gioco esperto. Se utilizzassimo al posto di un PC una workstation con 8 GPU potremmo arrivare a elaborare più di un miliardo di nodi al secondo, ma per raggiungere il livello di un grande maestro degli scacchi ci servirebbe ancora una funzione di valutazione ben regolata e un grande database di finali. I migliori programmi per gli scacchi, come STOCKFISH, dispongono di tutti questi elementi, e spesso raggiungono profondità di 30 livelli o più nell'albero di ricerca, superando di molto le capacità di qualsiasi giocatore umano.

5.3.4 Ricerca su albero e ricerca in tabelle

Sembrerebbe un compito esagerato, per un programma di scacchi, iniziare una partita considerando un albero di un miliardo di stati del gioco, solo per determinare la mossa di portare il pedone in e4 (la prima mossa più comune in assoluto). Da oltre un secolo esistono libri che trattano le migliori tattiche per l'apertura e la chiusura di una partita di scacchi (Tattersall, 1911). Non sorprende, quindi, il fatto che molti programmi di scacchi utilizzino ricerche in tabelle, invece che la ricerca su albero per aperture e chiusure.

Per le aperture, il computer si affida per lo più all'esperienza e alle competenze dell'uomo. I migliori consigli di grandi esperti su come effettuare ogni apertura possono essere copiati dai libri e inseriti in tabelle destinate all'uso da parte del computer. Inoltre, i computer possono raccogliere statistiche da un database di partite giocate nel passato per determinare quali sequenze di apertura portano più spesso a una vittoria. Per le prime mosse ci sono poche possibilità, e la maggior parte delle posizioni è già registrata nelle tabelle. Solitamente dopo 10 o 15 mosse ci si ritrova in una posizione incontrata raramente nel passato, e il programma deve passare dalla ricerca in tabelle alla ricerca su albero.

Verso la conclusione della partita ci sono ancora meno posizioni possibili, perciò è più facile cercare nelle tabelle. Ma qui è il computer l'esperto: l'analisi al computer delle chiusure supera di gran lunga le capacità umane. Un principiante umano può vincere un finale re-e-torre-contro-re (KRK) seguendo poche semplici regole, ma altri finali, come re, alfiere e cavallo contro re (KBNK), sono difficili da padroneggiare e non ammettono una strategia descrivibile in breve.

Un computer, invece, può risolvere interamente il finale di partita producendo una **politica** che associa a ogni possibile stato la migliore mossa in quello stato. La macchina, quindi, può giocare in modo perfetto cercando la mossa giusta in questa tabella. La tabella si costruisce mediante una ricerca minimax **retrograda**: si inizia considerando tutti i modi per disporre i pezzi KBNK sulla scacchiera; alcune posizioni corrisponderanno alla vittoria per il Bianco e andranno contrassegnate come tali. Poi si invertono le regole degli scacchi per fare “retromosse” anziché mosse reali. Qualsiasi mossa del Bianco che, a prescindere dalla mossa con cui risponda il Nero, termini in una posizione indicata come vittoria, deve anch'essa essere una vittoria. Si continua questa ricerca finché tutte le possibili posizioni sono risolte come vittorie, sconfitte o pareggi, ottenendo così una tabella infallibile per tutti i finali con i pezzi considerati. Ciò è stato fatto non solo per le chiusure KBNK, ma per tutti i finali con un massimo di sette pezzi, e le tabelle contengono 400 trilioni di posizioni. Una tabella per finali con otto pezzi dovrebbe contenere 40 quadrillioni di posizioni.

5.4 Ricerca ad albero Monte Carlo

Il gioco del Go illustra due importanti punti deboli della ricerca alfa-beta: in primo luogo, il Go ha un fattore di ramificazione che inizia da 361, il che significa che la ricerca alfa-beta dovrà essere limitata a soli 4 o 5 strati. In secondo luogo, è difficile definire una buona funzione di valutazione per il Go, perché il valore materiale non è un indicatore significativo e la maggior parte delle posizioni è fluida fino al termine del gioco. Per rispondere a queste due sfide, i programmi moderni per il Go hanno abbandonato la ricerca alfa-beta e usano invece una strategia chiamata **ricerca ad albero Monte Carlo** (*MCTS, Monte Carlo tree search*).³

La strategia di base per la ricerca ad albero Monte Carlo non utilizza una funzione di valutazione euristica, ma stima il valore di uno stato come utilità media su un certo numero di **simulazioni** di partite complete a iniziare dallo stato in questione. Una simulazione (detta anche **playout** o **rollout**) sceglie le mosse prima per un giocatore e poi per l'altro, quindi ripete fino a raggiungere una posizione terminale. A quel punto sono le regole del gioco (e non euristiche fallibili) a determinare chi ha vinto e chi ha perso, e con quale punteggio. Per i giochi in cui gli unici risultati possibili sono una vittoria o una sconfitta, “utilità media” è sinonimo di “percentuale di vittoria”.

Ora vi chiederete come si fa a scegliere quali mosse effettuare durante la simulazione. Se ci limitassimo a scegliere a caso, dopo un certo numero di simulazioni otterremmo una risposta alla domanda: “Qual è la mossa migliore se entrambi i giocatori giocano in modo casuale?”. Per alcuni giochi semplici la risposta a questa domanda è identica a quella che risponde a: “Qual è la mossa migliore se entrambi i giocatori giocano bene?”, ma per la maggior parte dei giochi non è così. Per ottenere informazioni utili dalla simulazione ci serve una **politica di simulazione** (*policy of playout*) che introduca una distorsione verso le mosse migliori. Per il Go e altri giochi, politiche di simulazione sono state apprese facendo giocare il programma contro se stesso e con l'uso di reti neurali. A volte si utilizzano euristiche specifiche per un gioco, come “considera le mosse di cattura” negli scacchi o “prendi la casella d'angolo” nel gioco Othello.

Una volta determinata una politica di simulazione, dobbiamo decidere due cose: da quali posizioni iniziare le simulazioni e quante simulazioni eseguire per ciascuna posizione. La risposta più semplice è la **ricerca Monte Carlo pura**, in cui si effettuano N simulazioni a partire dallo stato corrente del gioco e si registra quale delle mosse possibili nella posizione corrente ha la più alta percentuale di vittoria.

Per alcuni giochi stocastici si ha una convergenza al gioco ottimo al crescere di N , ma per la maggior parte dei giochi non è sufficiente, ci serve anche una **politica di selezione** che concentri selettivamente le risorse di calcolo su parti importanti dell'albero di gioco. Si bilanciano così due fattori: **esplorazione** (*exploration*) di stati per cui sono state effettuate poche simulazioni e **sfruttamento** (*exploitation*) di stati da cui si sono ottenuti buoni risultati in simulazioni passate per ottenere una stima più accurata dei loro valori (cfr. il Paragrafo 17.3 per ulteriori dettagli sul bilanciamento fra esplorazione/sfruttamento). La ricerca ad albero Monte Carlo mantiene a questo scopo un albero di ricerca che fa crescere a ogni iterazione con i quattro passi seguenti (Figura 5.10):

- **Selezione:** partendo dalla radice dell'albero di ricerca, scegliamo una mossa (guidati dalla politica di selezione) che porta a un nodo successore, poi ripetiamo il processo, spostandoci lungo l'albero fino a una foglia. La Figura 5.10(a) mostra un albero di ricerca con la radice che rappresenta uno stato in cui il Bianco ha appena mosso, e il Bianco ha vinto 37 delle 100 simulazioni effettuate finora. Una freccia più spessa indica la selezione, da parte del Nero, di una mossa che porta a un nodo dove il Nero ha vinto 60 simulazioni su 79. Questa

**ricerca ad albero
Monte Carlo**

**simulazione
playout
rollout**

**politica di
simulazione**

**ricerca Monte Carlo
pura**

politica di selezione

**esplorazione
sfruttamento**

³ Gli algoritmi “Monte Carlo” sono algoritmi randomizzati che prendono il nome dal casinò di Monte Carlo nel principato di Monaco.

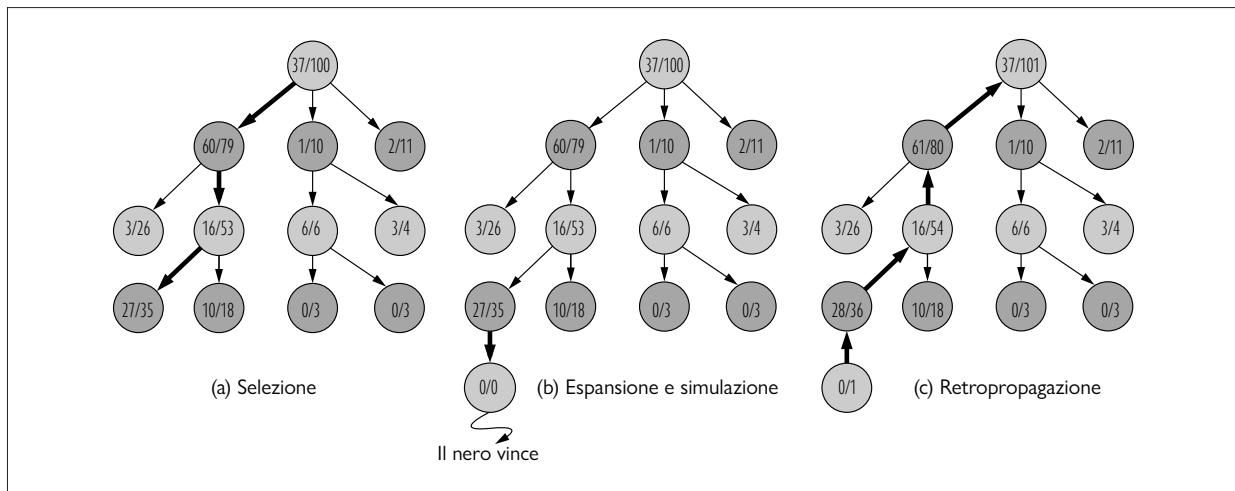


Figura 5.10 Una iterazione del processo di scegliere una mossa con la ricerca ad albero Monte Carlo, usando la politica di selezione UCT (limiti superiori di confidenza applicati agli alberi), mostrata dopo che sono già state effettuate 100 iterazioni. In (a) selezioniamo le mosse e percorriamo l'albero fino al nodo foglia indicato con 27/35 (che significa 27 vittorie per il Nero su 35 simulazioni). In (b) espandiamo il nodo selezionato ed effettuiamo una simulazione che termina in una vittoria per il Nero. In (c) i risultati della simulazione vengono retropropagati, risalendo l'albero fino alla radice.

è la migliore percentuale di vittoria fra le tre mosse, perciò la selezione di questa mossa è un esempio di sfruttamento. Ma sarebbe stato ragionevole anche selezionare il nodo 2/11 a scopo di esplorazione – con sole 11 simulazioni quel nodo ha ancora un alto grado di incertezza riguardo la valutazione, e potrebbe risultare migliore se ottenessimo ulteriori informazioni su di esso. La selezione continua fino al nodo foglia indicato con 27/35.

- **Espansione:** facciamo crescere l'albero di ricerca generando un nuovo figlio del nodo selezionato; nella Figura 5.10(b) il nuovo nodo è indicato con 0/0 (alcune versioni generano più figli in questo passaggio).
- **Simulazione:** eseguiamo una simulazione a partire dal nodo figlio appena generato, scegliendo le mosse per entrambi i giocatori in base alla politica di simulazione. Queste mosse *non* sono registrate nell'albero di ricerca. Nella figura, la simulazione porta a una vittoria per il Nero.
- **Retropropagazione:** ora utilizziamo il risultato della simulazione per aggiornare tutti i nodi dell'albero di ricerca, risalendo fino alla radice. Poiché il Nero ha vinto la simulazione, nei suoi nodi sono incrementati il numero di vittorie e il numero di simulazioni, perciò 27/35 diventa 28/36 e 60/79 diventa 61/80. Poiché il Bianco ha perso, nei suoi nodi viene incrementato soltanto il numero di simulazioni, perciò 16/53 diventa 16/54 e nella radice 37/100 diventa 37/101.

Ripetiamo questi quattro passaggi per un numero specifico di iterazioni, oppure fino allo scadere del tempo a disposizione, e poi restituiamo la mossa con il più alto numero di simulazioni.

UCT

Una politica di selezione molto efficace è quella denominata **UCT** (*upper confidence bounds applied to trees*, letteralmente “limiti superiori di confidenza applicati ad alberi”), che classifica ogni possibile mossa in base a una formula per il limite superiore di confidenza denominata **UCB1** (cfr. il Paragrafo 17.3.3 per ulteriori dettagli). Per un nodo n , la formula è:

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PADRE(n))}{N(n)}}$$

UCB1

dove $U(n)$ è l'utilità totale di tutte le simulazioni che sono passate per il nodo n , $N(n)$ è il numero di simulazioni passate per il nodo n , $\text{PADRE}(n)$ è il nodo padre di n nell'albero. Quindi $\frac{U(n)}{N(n)}$ è il termine di sfruttamento: l'utilità media di n . Il termine con la radice quadrata è il termine di esplorazione: al denominatore c'è il conteggio $N(n)$, per cui il termine sarà alto per nodi che sono stati esplorati solo poche volte; al numeratore c'è il logaritmo del numero di esplorazioni del nodo genitore di n : questo significa che, se selezioniamo n per una percentuale non nulla del tempo, il termine di esplorazione tende a zero all'aumentare del numero di simulazioni e alla fine si arriverà al nodo con l'utilità media più alta.

C è una costante che bilancia sfruttamento ed esplorazione. Secondo un'argomentazione teorica C dovrebbe essere $\sqrt{2}$, ma nella pratica i programmati di giochi sperimentano più valori per C e scelgono quello che offre le prestazioni migliori (alcuni programmi usano formule leggermente diverse; per esempio **ALPHAZERO** aggiunge un termine per la probabilità della mossa, che viene calcolata da una rete neurale addestrata in base a partite giocate in passato). Con $C = 1,4$ è il nodo 60/79 nella Figura 5.10 che ha il punteggio UCB1 più alto, ma con $C = 1,5$ sarebbe il nodo 2/11.

La Figura 5.11 mostra l'algoritmo di ricerca ad albero Monte Carlo UCT completo. Quando le iterazioni terminano, viene restituita la mossa con il più alto numero di simulazioni. Si potrebbe pensare che sarebbe meglio restituire il nodo con l'utilità media più alta, ma l'idea è che un nodo con 65/100 vittorie è migliore di uno con 2/3 vittorie, perché il secondo presenta molta incertezza. In ogni caso, la formula UCB1 garantisce che il nodo con più simulazioni sia quasi sempre quello con la più alta percentuale di vittoria, perché il processo di selezione favorisce sempre di più la percentuale di vittoria all'aumentare del numero di simulazioni.

Il tempo richiesto per il calcolo di una simulazione è lineare, non esponenziale, nella profondità dell'albero del gioco, perché a ogni punto di scelta si effettua solo una mossa. Abbiamo quindi parecchio tempo da usare per simulazioni multiple. Per esempio, consideriamo un gioco con fattore di ramificazione 32, in cui la partita media si svolge su 100 strati. Se avessimo potenza di calcolo sufficiente per considerare un miliardo di stati prima di fare una mossa, l'algoritmo minimax potrebbe cercare fino a una profondità di 6 strati, mentre l'algoritmo alfa-beta con ordinamento delle mosse perfetto potrebbe arrivare a 12 strati, e la ricerca Monte Carlo potrebbe effettuare 10 milioni di simulazioni. Quale sarà l'approccio migliore? Dipende dall'accuratezza della funzione euristica rispetto alle politiche di selezione e simulazione.

Per opinione diffusa si ritiene che la ricerca Monte Carlo sia più vantaggiosa rispetto alla ricerca alfa-beta per giochi come il Go in cui il fattore di ramificazione è molto alto (e quindi la ricerca alfa-beta non può arrivare abbastanza in profondità), o quando è difficile definire

```

function RICERCA-ALBERO-MONTE-CARLO(stato) returns un'azione
    albero  $\leftarrow$  NODO(stato)
    while RIMANE-TEMPO() do
        foglia  $\leftarrow$  SELEZIONA(albero)
        figlio  $\leftarrow$  ESPANDI(foglia)
        risultato  $\leftarrow$  SIMULA(figlio)
        RETROPROPAGA(risultato, figlio)
    return la mossa in AZIONI(stato) il cui nodo ha il più alto numero di simulazioni

```

Figura 5.11 L'algoritmo per la ricerca ad albero Monte Carlo. Per prima cosa viene inizializzato un albero del gioco, denominato *albero*, e poi si ripete un ciclo con SELEZIONA / ESPANDI / SIMULA / RETROPROPAGA finché si esaurisce il tempo, restituendo infine la mossa che ha portato al nodo con il più alto numero di simulazioni.

terminazione anticipata della simulazione

una buona funzione di valutazione. La ricerca alfa-beta sceglie il cammino per arrivare a un nodo che ha il più alto punteggio raggiungibile in base a una funzione di valutazione, dato che l'avversario cercherà di minimizzare il punteggio. Quindi, se la funzione di valutazione è imprecisa, la ricerca alfa-beta sarà anch'essa imprecisa. Un calcolo sbagliato su un singolo nodo può portare la ricerca alfa-beta a un errore nella scelta di intraprendere un cammino per raggiungere quel nodo (o per evitarlo). La ricerca Monte Carlo, invece, si basa sull'aggregato di molte simulazioni, quindi non è vulnerabile a un singolo errore. È possibile combinare ricerca Monte Carlo e funzioni di valutazione eseguendo una simulazione per un certo numero di mosse e interrompendola per applicare una funzione di valutazione.

È anche possibile combinare elementi delle ricerche alfa-beta e Monte Carlo. Per esempio, nei giochi in cui le mosse possono essere numerose potrebbe essere utile ricorrere alla **terminazione anticipata della simulazione**: interrompiamo una simulazione che sta eseguendo troppe mosse e la valutiamo con una funzione euristica oppure dichiariamo la parità.

La ricerca Monte Carlo può essere applicata anche a giochi del tutto nuovi, in cui non esiste un corpo di esperienze su cui basarsi per definire una funzione di valutazione. Per una ricerca Monte Carlo tutto ciò che serve davvero è la conoscenza delle regole del gioco. Le politiche di selezione e simulazione possono fare buon uso delle conoscenze acquisite, quando sono disponibili, ma è anche possibile apprendere buone politiche usando reti neurali addestrate facendo giocare il programma contro se stesso.

Quando è probabile che una singola mossa possa cambiare il corso del gioco, una ricerca Monte Carlo ha uno svantaggio dovuto alla sua natura stocastica, per cui potrebbe trascurare tale mossa. In altre parole, con la potatura di Tipo B nella ricerca Monte Carlo, c'è il rischio che una linea di gioco vitale possa non essere mai esplorata. Un altro svantaggio della ricerca Monte Carlo si ha quando ci sono stati del gioco che rappresentano “ovviamente” una vittoria per uno dei giocatori (secondo la conoscenza umana e una funzione di valutazione), ma per verificarlo servirebbero ancora molte mosse. A lungo si è ritenuto che la ricerca alfa-beta fosse più adatta a giochi come gli scacchi con basso fattore di ramificazione e buone funzioni di valutazione, ma recentemente il metodo Monte Carlo ha dimostrato il suo valore anche negli scacchi e in altri giochi.

Il concetto generale di simulare mosse future, osservarne i risultati e usarli per determinare quali sono le buone mosse è un tipo di **apprendimento con rinforzo**, argomento trattato nel Capitolo 22 del Volume 2.

5.5 Giochi stocastici

gioco stocastico

I **giochi stocastici** ci portano un po' più vicino all'imprevedibilità della vita reale includendo un elemento casuale, come il lancio di un dado. Il backgammon è un tipico gioco stocastico che combina fortuna e abilità. Nella posizione della Figura 5.12, il Bianco ha appena tirato 6–5 e ha quattro mosse possibili (ognuna delle quali muove un pezzo in avanti in senso orario di 5 posizioni e uno in avanti di 6 posizioni).

nodi di casualità

A questo punto il Nero sa quali mosse si possono fare, ma non sa il risultato dei dadi del Bianco e quindi neppure le mosse che potrà fare. Questo significa che il Nero non può costruire un albero di gioco standard come quelli che abbiamo visto per gli scacchi e il gioco del tris. Oltre ai nodi MAX e MIN, un albero di gioco per il backgammon deve includere **nodi di casualità** (che nella Figura 5.13 sono rappresentati come cerchi). I rami uscenti da ogni nodo di casualità rappresentano i diversi esiti del lancio dei dadi; ognuno è etichettato con la configurazione dei dadi e la rispettiva probabilità. Ci sono 36 modi di tirare due dadi, tutti equiprobabili; dato però che ai fini del gioco 6–5 è lo stesso di 5–6, i possibili tiri distinti diventano 21. Ogni tiro doppio (da 1–1 a 6–6) ha una probabilità di verificarsi di 1/36, gli altri 15 tiri hanno ognuno una probabilità di 1/18.

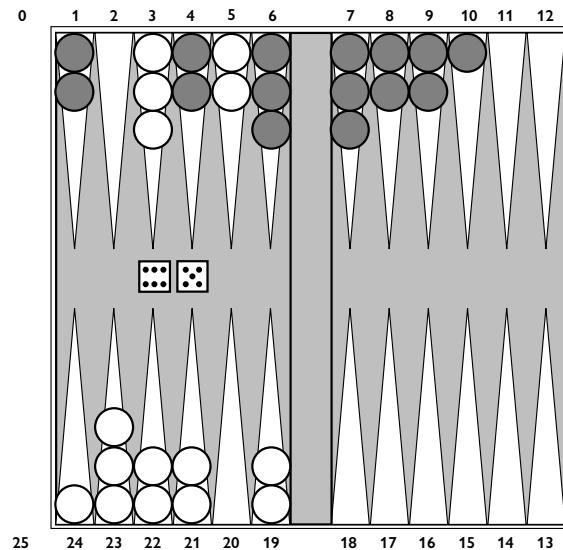


Figura 5.12 Una tipica posizione del backgammon: lo scopo del gioco è far uscire tutti i propri pezzi dal tavoliere. Il Bianco muove in senso orario verso la posizione indicata dal numero 25, il Nero in senso antiorario verso lo 0. Un pezzo può muoversi in qualsiasi posizione, a meno che questa non contenga due o più pezzi avversari; se ce n'è uno solo viene catturato e deve ricominciare dall'inizio. Nella posizione indicata, il Bianco ha tirato 6–5 e deve scegliere tra quattro mosse legali: (5–11, 5–10), (5–11, 19–24), (5–10, 10–16) e (5–11, 11–16), dove la notazione (5–11, 11–16) significa di muovere un pezzo dalla posizione 5 alla 11 e poi muovere un pezzo dalla 11 alla 16.

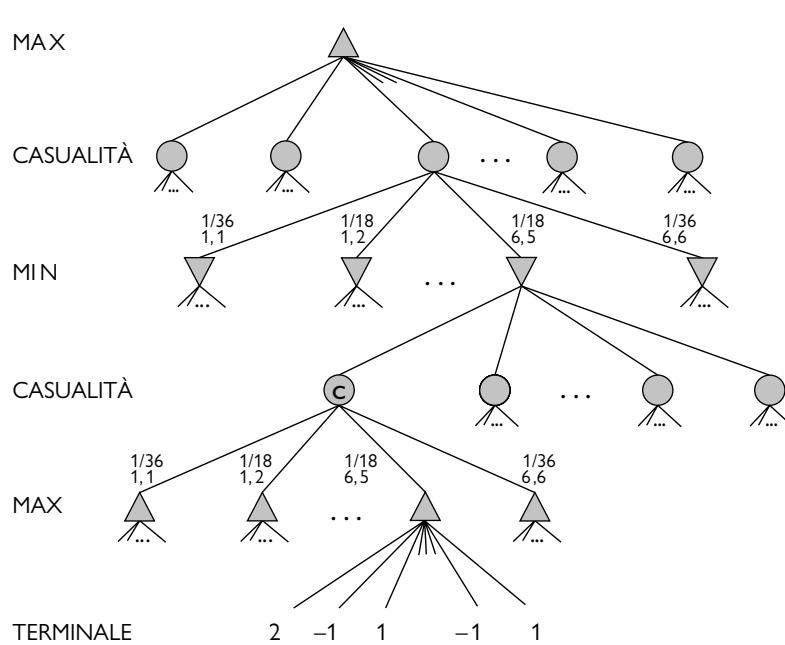


Figura 5.13

Uno schema di albero di gioco per il backgammon.

valore atteso

Il passo successivo è capire come prendere le decisioni corrette. Naturalmente vogliamo ancora scegliere la mossa che porta alla posizione migliore. Ora però le posizioni non hanno più valori minimax definiti: al loro posto possiamo solo calcolare il **valore atteso**, cioè la media su tutti i possibili risultati dei nodi di casualità.

valore expectiminimax

Questo ci porta al **valore expectiminimax** per i giochi con nodi di casualità, una generalizzazione del valore minimax per i giochi deterministici. I nodi terminali, quelli MAX e quelli MIN funzionano esattamente come prima (tenendo presente che le mosse legali per i nodi MAX e MIN dipenderanno dal risultato dei dadi gettati nel nodo di casualità precedente). Per i nodi di casualità si calcola il valore atteso, ovvero la somma del valore di tutti i risultati pesati in base alla probabilità di ciascuna azione:

$$\text{EXPECTIMINIMAX}(n) =$$

$$\begin{cases} \text{UTILITÀ}(s) & \text{se } \dot{\text{E}}\text{-TERMINALE}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RISULTATO}(s, a)) & \text{se } \text{DEVE-MUOVERE}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RISULTATO}(s, a)) & \text{se } \text{DEVE-MUOVERE}(s) = \text{MIN} \\ \sum_r P(r) \text{ EXPECTIMINIMAX}(\text{RISULTATO}(s, r)) & \text{se } \text{DEVE-MUOVERE}(s) = \text{CASUALITÀ} \end{cases}$$

dove r rappresenta un possibile lancio di dadi (o un altro evento possibile) e $\text{RISULTATO}(s, r)$ è lo stesso stato di s , con in più il fatto che il risultato del tiro di dadi è r .

5.5.1 Funzioni di valutazione per giochi con elementi casuali

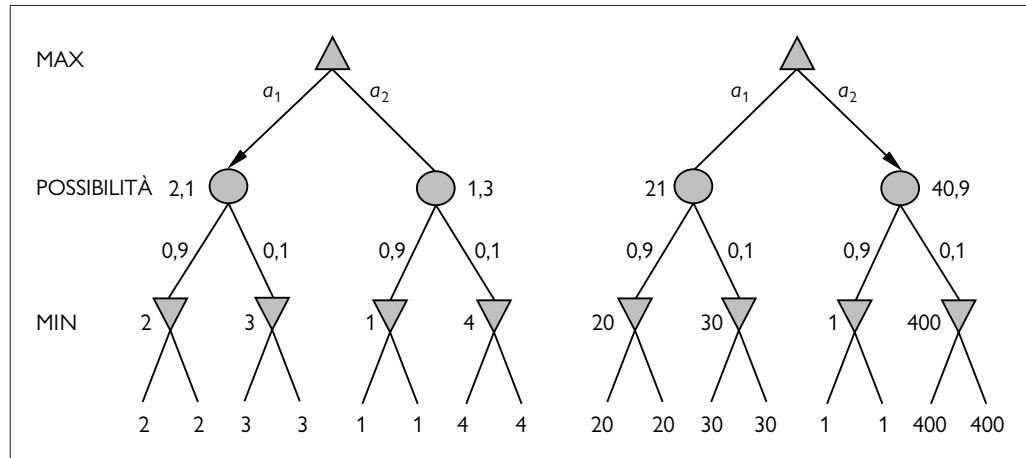
Come nel caso del minimax, l'approssimazione più ovvia con expectiminimax è tagliare la ricerca a un certo livello e applicare una funzione di valutazione a ogni foglia. Si potrebbe pensare che le funzioni di valutazione per giochi come il backgammon siano analoghe a quelle degli scacchi: il loro compito sarebbe quindi semplicemente quello di assegnare valori più alti alle posizioni migliori. In realtà, la presenza dei nodi di casualità modifica lo stesso significato dei valori.

La Figura 5.14 illustra ciò che accade: con una funzione di valutazione che assegna alle foglie i valori $[1, 2, 3, 4]$, la mossa migliore è a_1 ; con i valori $[1, 20, 30, 400]$, la mossa migliore diventa a_2 . Quindi il programma si comporta in modo totalmente diverso se cambiamo alcuni dei valori della valutazione, anche se l'ordine di preferenza rimane lo stesso.

Per evitare questo problema, la funzione di valutazione deve restituire valori che siano una trasformazione lineare positiva della **probabilità** di vincere (o dell'utilità attesa, per gio-

Figura 5.14

Una trasformazione dei valori delle foglie che ne preserva l'ordinamento modifica tuttavia la scelta della mossa migliore.



chi che hanno risultati diversi da vittoria/sconfitta). Questa relazione con la probabilità è un'importante proprietà generale delle situazioni di incertezza, e la esamineremo meglio nel Capitolo 16.

Se il programma conoscesse in anticipo tutti i tiri di dado di una partita, una soluzione per i giochi che utilizzano i dadi si potrebbe ottenere allo stesso modo che per quelli che non hanno elementi casuali, cosa che minimax fa in un tempo $O(b^m)$, dove b è il fattore di ramificazione e m è la profondità massima dell'albero. Dato che expectiminimax considera anche tutte le possibili sequenze di tiri, la complessità sale a $O(b^m n^m)$, dove n è il numero di tiri di dado distinti.

Anche se la profondità della ricerca viene limitata a un livello d abbastanza piccolo, il costo aggiuntivo rispetto a quello di minimax rende poco realistica la possibilità di guardare molto avanti nei giochi con elementi casuali. Nel backgammon n vale 21 e b solitamente è intorno a 20, ma in alcune situazioni può anche arrivare a 4000 quando si ottiene lo stesso numero con entrambi i dadi. Con tutta probabilità, non sarà possibile gestire più di tre strati dell'albero.

Un altro modo di esprimere questo concetto è il seguente: il vantaggio della ricerca alfa-beta è che permette di ignorare sviluppi futuri che, dato uno svolgimento ottimo della partita, sono semplicemente impossibili; in questo modo la ricerca si può concentrare sulle mosse future più probabili. Tuttavia, in un gioco in cui vi è un tiro di dadi prima di ogni mossa, *non esistono* sequenze di mosse probabili; anche la mossa più probabile viene fatta soltanto 2/36 volte, perché è il tiro dei dadi che determina quale mossa viene fatta. Questo è un problema generale ogni volta che entra in gioco l'incertezza: le possibilità si moltiplicano enormemente e diventa inutile formulare piani d'azione dettagliati, perché il mondo li renderà con ogni probabilità irrealizzabili.

Qualche lettore avrà pensato che forse è possibile applicare tecniche come la potatura alfa-beta agli alberi di gioco con nodi di casualità: in effetti si può. Per i nodi MIN e MAX l'analisi non cambia, ma con un po' d'ingegno possiamo potare anche qualche nodo di casualità. Considerate il nodo C nella Figura 5.13 e quello che succede al suo valore quando valutiamo i suoi figli. Ricordando che questo è ciò che serve alla tecnica alfa-beta per potare un sotto-albero, è possibile trovare un limite superiore al valore di C prima di considerare tutti i suoi nodi figli? A prima vista potrebbe sembrare impossibile, perché il valore di C è la *media* del valore dei nodi figli, e per calcolare la media di un insieme di numeri, è necessario considerarli tutti. Ma se limitiamo i possibili valori della funzione di utilità, possiamo derivare dei limiti anche per la media senza esaminare ogni singolo numero. Per esempio, se diciamo che i valori di utilità sono sempre compresi tra -2 e +2, il valore delle foglie risulta vincolato, e di conseguenza *possiamo* derivare un limite superiore al valore di un nodo di casualità senza considerare tutti i suoi figli.

Nei giochi in cui il fattore di ramificazione per i nodi di casualità è alto, come per esempio lo Yahtzee in cui si tirano 5 dadi a ogni turno, può essere utile considerare una potatura in avanti che effettui un campionamento di un piccolo numero dei possibili rami casuali. Oppure sarebbe preferibile evitare del tutto l'uso di una funzione di valutazione e optare invece per una ricerca ad albero Monte Carlo, in cui ogni simulazione include tiri di dadi casuali.

5.6 Giochi parzialmente osservabili

Bobby Fisher disse che “il gioco degli scacchi è una guerra”, ma in realtà manca di almeno una fondamentale caratteristica delle guerre reali: l'**osservabilità parziale**. Nella “nebbia della guerra”, il posizionamento delle unità nemiche è spesso ignoto, finché non si rivela per contatto diretto. Di conseguenza, l’attività di guerra comprende l’utilizzo di esploratori e spie per raccogliere informazioni e il ricorso a operazioni di occultamento e bluff per confondere il nemico.

Kriegspiel

I giochi parzialmente osservabili condividono queste caratteristiche e sono quindi diversi, qualitativamente, da quelli descritti nei paragrafi precedenti. I videogiochi come StarCraft sono particolarmente difficili da affrontare, dato che sono parzialmente osservabili, multigente, dinamici e ignoti.

Nei giochi parzialmente osservabili *deterministici*, l'incertezza sullo stato del gioco nasce interamente dall'impossibilità di accedere alle scelte fatte dall'avversario. Questa classe comprende giochi per bambini come Battaglia navale (dove le navi di ciascun giocatore sono poste in punti nascosti all'avversario) e Stratego (dove le posizioni dei pezzi sono note, ma sono nascosti i tipi di pezzi). Esamineremo il gioco **Kriegspiel**, una variante parzialmente osservabile degli scacchi in cui i pezzi possono muoversi ma sono completamente invisibili all'avversario. Tra gli altri giochi con versioni parzialmente osservabili vi sono Phantom Go, Phantom tic-tac-toe e Screen Shogi.

5.6.1 Kriegspiel: scacchi parzialmente osservabili

Le regole del Kriegspiel sono le seguenti: Bianco e Nero vedono ciascuno una scacchiera contenente soltanto i loro pezzi. Un arbitro, che può vedere tutti i pezzi, controlla la partita e periodicamente fa degli annunci che entrambi i giocatori possono ascoltare. Per prima cosa, il Bianco propone all'arbitro una mossa che sarebbe legale se non vi fossero pezzi neri; se i pezzi del Nero impediscono la mossa, l'arbitro annuncia: “Illegal” e il Bianco può continuare a proporre mosse finché ne trova una legale (intanto apprende informazioni sulla posizione dei pezzi neri).

Quando viene proposta una mossa legale, l'arbitro fa uno o più dei seguenti annunci: “Cattura sulla casella X ” se c'è una cattura, “Scacco da D ” se il re nero è sotto scacco, dove D è la direzione dello scacco e può essere “Cavallo”, “Riga”, “Colonna”, “Diagonale lunga” o “Diagonale corta”. Se il Nero è in scacco matto o in stallo, l'arbitro lo dice; altrimenti, tocca al Nero muovere.

Questo gioco potrebbe sembrare impossibile, ma gli uomini lo giocano abbastanza bene e i programmi per computer stanno mettendosi al passo. È utile richiamare il concetto di **stato-credenza** definito nel Paragrafo 4.4 e illustrato nella Figura 4.14: è l'insieme di tutti gli stati del gioco *logicamente possibili* data la storia completa delle percezioni passate. Inizialmente, lo stato-credenza del Bianco è un singoletto, perché i pezzi del Nero non sono ancora stati mossi. Dopo che il Bianco fa una mossa e il Nero risponde, lo stato-credenza del Bianco contiene 20 posizioni, perché il Nero ha 20 modi per rispondere a qualsiasi mossa di apertura. Tenere traccia dello stato-credenza durante il procedere del gioco è esattamente il problema della **stima dello stato**, per cui l'Equazione (4.6) nel Paragrafo 4.4.4 fornisce il passo di aggiornamento. Possiamo far corrispondere direttamente la stima dello stato del gioco Kriegspiel alla struttura non deterministica e parzialmente osservabile del Paragrafo 4.4, se consideriamo l'avversario come la fonte del non determinismo; ovvero, i **RISULTATI** del Bianco sono costituiti dai risultati (prevedibili) delle mosse del Bianco stesso e dai risultati imprevedibili forniti dalle risposte del Nero.⁴

Dato uno stato-credenza corrente, il Bianco potrebbe chiedere: “Posso vincere la partita?”. In un gioco parzialmente osservabile, il concetto di **strategia** è alterato: invece di specificare una mossa da compiere per ogni possibile *mossa* che l'avversario potrebbe fare, ci serve una mossa per ogni possibile sequenza di percezioni che potrebbe essere ricevuta. Nel caso del Kriegspiel, una strategia vincente, o **scacco matto garantito**, è tale che, per ogni possibile sequenza di percezioni, porta a uno scacco matto per ogni possibile stato della scac-

scacco matto garantito

⁴ Talvolta lo stato-credenza diventerà troppo grande per rappresentare anche solo un elenco di stati della scacchiera, ma per ora ignoreremo questo aspetto. Nei Capitoli 7 e 8 sono suggeriti dei metodi per rappresentare in modo compatto stati-credenza molto grandi.

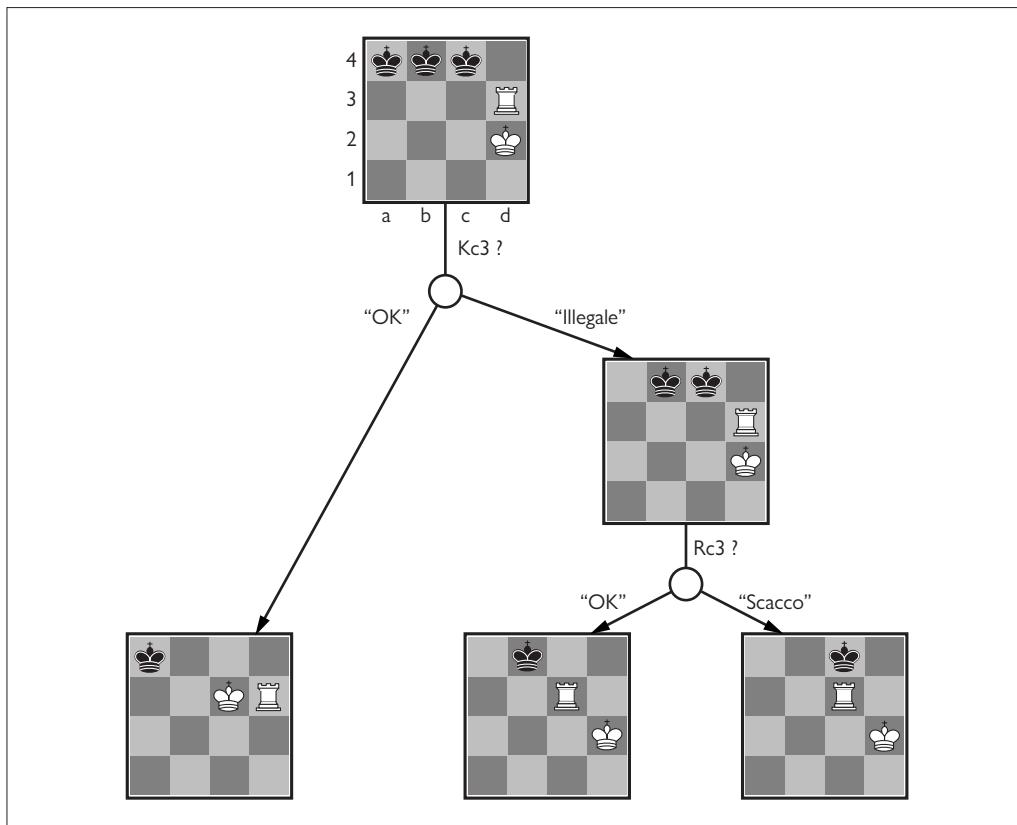


Figura 5.15
Parte di una strategia di scacco matto garantito nella chiusura KRK, mostrata su una scacchiera ridotta. Nello stato-credenza iniziale, il re nero può essere in una posizione fra tre possibili. Con una combinazione di mosse, la strategia restringe le possibilità a una. Lasciamo come esercizio il completamento dello scacco matto.

chiera nello stato-credenza corrente, a prescindere da come muove l'avversario. Con questa definizione, lo stato-credenza dell'avversario è irrilevante, la strategia deve funzionare anche se l'avversario può vedere tutti i pezzi. Questo semplifica notevolmente i calcoli. La Figura 5.15 mostra parte di una strategia di scacco matto garantito per la chiusura KRK (re e torre contro re). In questo caso il Nero ha soltanto un pezzo (il re), perciò uno stato-credenza per il Bianco può essere mostrato in una sola scacchiera contrassegnando ogni possibile posizione del re nero.

L'algoritmo generale di ricerca AND-OR può essere applicato allo spazio degli stati-credenza per trovare strategie di scacco matto garantito, come si è visto nel Paragrafo 4.4. L'algoritmo stato-credenza incrementale citato nel Paragrafo 4.4.1 spesso trova uno scacco matto in situazioni a metà di una partita fino alla profondità 9, ben oltre le capacità dei giocatori umani.

Oltre alle strategie di scacco matto garantito, Kriegspiel ammette un concetto del tutto nuovo che non ha senso in giochi completamente osservabili: lo **scacco matto probabilistico**. Tali strategie di scacco matto devono sempre funzionare in ogni stato della scacchiera compreso nello stato-credenza e sono probabilistiche rispetto alla randomizzazione delle mosse del giocatore vincente. Per capire l'idea di base, considerate il problema di trovare un re nero utilizzando soltanto il re bianco. Muovendo a caso, il re bianco *alla fine* raggiungerà il nero anche se quest'ultimo tenta di evitarlo, perché il Nero non può continuare indefinitamente a indovinare le mosse evasive giuste. Nella terminologia della teoria della probabilità, l'evento di localizzazione si verifica *con probabilità 1*.

La chiusura KBNK (re, alfiere e cavallo contro re) in questo senso è vittoriosa: il Bianco costringe il Nero a una sequenza casuale infinita di scelte, per una delle quali il Nero sba-

scacco matto probabilistico



glierà a indovinare la mossa del Bianco e rivelerà la sua posizione portando allo scacco matto. La chiusura KBBK (re, alfiere e alfiere contro re), invece, è vittoriosa con probabilità $1 - \varepsilon$. Il Bianco può forzare una vittoria soltanto lasciando uno dei suoi alfieri privo di protezione per una mossa. Se il Nero si trova nel posto giusto e cattura l'alfiere (una mossa che sarebbe illegale se gli alfieri fossero protetti), la partita sarà in parità. Il Bianco può scegliere di fare questa mossa rischiosa in un punto scelto a caso di una sequenza molto lunga, riducendo così ε a una costante arbitrariamente piccola, ma non può ridurre ε a zero.

A volte una strategia di scacco matto funziona per *alcuni* degli stati della scacchiera nello stato-credenza attuale, ma non per altri. Provare una simile strategia potrebbe portare al successo, con uno **scacco matto accidentale** (accidentale nel senso che il Bianco potrebbe non *sapere* che otterrebbe lo scacco) se i pezzi del Nero si trovano nelle posizioni giuste (la maggior parte degli scacchi matti nelle partite tra uomini sono di natura accidentale in questo senso). Questa idea porta naturalmente a chiedersi *quale probabilità* vi sia che una data strategia risulti vittoriosa, il che porta a sua volta a chiedersi *quale probabilità* vi sia che ogni stato della scacchiera nello stato-credenza corrente sia lo stato vero.

Il primo orientamento potrebbe essere quello di proporre che tutti gli stati della scacchiera nello stato-credenza corrente siano equiprobabili, ma non può essere così. Consideriamo, per esempio, lo stato-credenza del Bianco dopo la prima mossa del Nero nella partita. Per definizione (supponendo che il Nero giochi in modo ottimo) il Nero ha fatto una mossa ottima, perciò tutti gli stati della scacchiera risultanti da mosse subottime dovrebbero avere probabilità nulla. Anche questo argomento non è del tutto corretto, perché *lo scopo di ciascun giocatore non è solo di muovere i pezzi nelle caselle giuste, ma anche di ridurre al minimo le informazioni in possesso dell'avversario sulla loro posizione*. Giocando qualsiasi strategia “ottima” *prevedibile* si forniscono informazioni all'avversario. Quindi, una strategia ottima nei giochi parzialmente osservabili richiede la disponibilità a giocare anche *in modo casuale* (ecco perché gli ispettori dell'ufficio igiene fanno ispezioni *a caso* nei ristoranti). Questo significa scegliere talvolta mosse che potrebbero sembrare “intrinsicamente” deboli, ma che traggono la loro forza dalla loro imprevedibilità, perché è improbabile che l'avversario abbia predisposto una difesa contro di esse.

In base a queste considerazioni sembra che le probabilità associate agli stati della scacchiera nello stato-credenza corrente possano soltanto essere calcolate data una strategia ottima randomizzata; a sua volta, il calcolo di questa strategia sembra richiedere di conoscere le probabilità dei vari stati in cui potrebbe trovarsi la scacchiera. Questo rompicapo può essere risolto adottando il concetto della teoria dei giochi noto come **soluzione di equilibrio**, che tratteremo meglio nel Capitolo 17. Un equilibrio specifica una strategia randomizzata ottima per ciascun giocatore. Il calcolo degli equilibri è troppo costoso per il Kriegspiel. Al giorno d'oggi, la progettazione di algoritmi efficaci per partite a Kriegspiel è un argomento di ricerca aperto. La maggior parte dei sistemi svolge una ricerca in avanti con profondità limitata nel proprio spazio degli stati-credenza, ignorando quello dell'avversario. Le funzioni di valutazione somigliano a quelle per il gioco osservabile, ma includono una componente per la dimensione dello stato-credenza: piccolo è bello! Torneremo ai giochi parzialmente osservabili quando tratteremo la teoria dei giochi nel Paragrafo 18.2.

5.6.2 Giochi di carte

I giochi di carte come bridge, whist, hearts (noto in alcune regioni italiane come “la peppa”) e poker sono caratterizzati da osservabilità parziale *stocastica*, in cui le informazioni mancanti sono generate da una distribuzione casuale delle carte.

A prima vista potrebbe sembrare che i giochi di carte siano molto simili a quelli che usano i dadi: le carte sono distribuite casualmente e determinano le mosse consentite a ogni giocatore, ma è come se all'inizio della partita i dadi siano già stati tirati tutti! Questa analogia

risulta non corretta, ma suggerisce un algoritmo: considerare l'inizio del gioco come un nodo di causalità con ogni possibile distribuzione delle carte, e poi usare la formula EXPECTIMINIMAX per scegliere la mossa migliore. Notate che in questo approccio l'unico nodo di causalità è la radice, poi il gioco diventa completamente osservabile. Talvolta si parla di *media sulla chiaroveggenza* perché si assume che, una volta fatta la distribuzione, il gioco diventi completamente osservabile per entrambi i giocatori. Nonostante il suo appeal intuitivo, la strategia può portare fuori strada. Considerate il seguente racconto:

Giorno 1: la Strada A porta a una pentola d'oro; la Strada B porta a un bivio. Potete vedere che la diramazione di sinistra porta a due pentole d'oro, mentre seguendo quella di destra sarete investiti da un autobus.

Giorno 2: la Strada A porta a una pentola d'oro; la Strada B porta a un bivio. Potete vedere che la diramazione di destra porta a due pentole d'oro, mentre seguendo quella di sinistra sarete investiti da un autobus.

Giorno 3: la Strada A porta a una pentola d'oro; la Strada B porta a un bivio. Vi si dice che una diramazione porta a due pentole d'oro mentre seguendo l'altra sarete investiti da un autobus. Sfortunatamente non sapete qual è la via giusta.

La media sulla chiaroveggenza porta al seguente ragionamento: al giorno 1, la scelta giusta è B; al giorno 2, la scelta giusta è B; al giorno 3, la situazione è identica al giorno 1 o al giorno 2, perciò B deve essere ancora la scelta giusta.

Ora possiamo vedere perché la strategia della media sulla chiaroveggenza fallisce: non considera lo *stato-credenza* in cui l'agente si troverà dopo l'azione. Uno stato-credenza di totale ignoranza non è desiderabile, soprattutto quando una possibilità è la morte certa. Poiché questo approccio ipotizza che ogni stato futuro sarà automaticamente di conoscenza perfetta, non sceglie mai azioni che *ottengono informazioni* (come la prima mossa nella Figura 5.15), né azioni che nascondono informazioni all'avversario o le forniscono a un compagno, perché ipotizza che essi conoscano già le informazioni. Inoltre, con questo approccio un giocatore a poker non farà mai un **bluff**,⁵ perché ipotizza che l'avversario possa vedere le sue carte. Nel Capitolo 17 mostreremo come costruire algoritmi che facciano tutte queste cose risolvendo il vero problema decisionale parzialmente osservabile, ottenendo una strategia di equilibrio ottimo (Paragrafo 18.2).

bluff

Nonostante tutti i problemi, la strategia della media sulla chiaroveggenza può essere efficace, ed esistono alcuni trucchi per farla funzionare ancora meglio. Nella maggior parte dei giochi di carte, il numero delle possibili distribuzioni è piuttosto grande. Per esempio, nel bridge ogni giocatore vede solo due delle quattro mani; ci sono due mani nascoste di 13 carte ciascuna, perciò il numero di distribuzioni delle carte è $\binom{26}{13} = 10,400,600$. Risolvere anche una sola distribuzione è piuttosto difficile, perciò risolverne dieci milioni è fuori questione. In questi casi si può ricorrere all'**astrazione**, considerando identiche le mani simili. Per esempio, è molto importante sapere quali assi e re ci sono in una mano, ma i 4 e i 5 contano poco e possono essere ignorati.

Un altro modo per affrontare numeri molto grandi è la potatura in avanti: consideriamo solo un piccolo campione casuale di N distribuzioni e calcoliamo ancora il punteggio EXPECTIMINIMAX. Anche per N abbastanza piccolo, diciamo da 100 a 1000, questo metodo fornisce una buona approssimazione, inoltre può anche essere applicato a giochi deterministici

⁵ Bluffare, ovvero puntare come se si avesse una buona mano, quando invece non la si ha, è una parte fondamentale della strategia nel poker.

quali Kriegspiel, dove si può considerare un campione dei possibili stati del gioco anziché possibili distribuzioni di carte, purché si disponga di un modo per stimare la probabilità di ogni stato. Può essere utile anche effettuare una ricerca euristica con una soglia di profondità, anziché sull'intero albero di gioco.

Finora abbiamo ipotizzato che tutte le distribuzioni di carte fossero equiprobabili, cosa sensata per giochi come whist e hearts. Nel caso del bridge, invece, la partita è preceduta da una fase di “dichiarazione” in cui ogni squadra indica quante prese si aspetta di vincere. Poiché i giocatori fanno la dichiarazione in base alle carte che hanno, gli altri giocatori ottengono informazioni sulla probabilità $P(s)$ di ogni distribuzione. Tenere conto di ciò nel decidere come giocare la mano è difficile, per le ragioni citate precedentemente nella descrizione del gioco Kriegspiel: i giocatori potrebbero fare le dichiarazioni in modo da ridurre al minimo le informazioni fornite ai loro avversari.

Nel gioco del poker i computer hanno raggiunto prestazioni superumane (o superiori a quelle degli umani). Il programma Libratus ha affrontato quattro dei migliori giocatori di poker al mondo in una sfida di 20 giorni nella specialità Texas hold'em senza limiti, e li ha battuti tutti. Dato che nel poker gli stati possibili sono numerosissimi, Libratus ricorre all'astrazione per ridurne il numero: potrebbe considerare equivalenti le due mani AAA72 e AAA64 (sono entrambe “tre assi e carte basse”) e anche considerare equivalenti una puntata da 200 euro e una da 201 euro. Il programma inoltre controlla gli altri giocatori e, se rileva che anche loro sfruttano una astrazione, esegue altri calcoli per chiudere la falla nelle partite successive. Libratus ha usato complessivamente 25 milioni di ore della CPU di un supercomputer, per arrivare alla vittoria.

Il costi computazionali sostenuti da Libratus (e costi simili richiesti da ALPHAZERO e altri sistemi) indicano che arrivare a giocare al livello del campione del mondo potrebbe essere fuori portata per i ricercatori con budget limitato. E in parte questo è vero: avere la possibilità di accedere a un supercomputer o a componenti hardware come i TPU (*tensor processing unit*) fornisce certamente un vantaggio, e questo è particolarmente vero per l’addestramento di un sistema, tuttavia l’addestramento può anche essere effettuato attraverso il crowdsourcing. Per esempio, il sistema open source LEELAZERO è una reimplementazione di ALPHAZERO che esegue l’addestramento attraverso il gioco contro se stesso sui computer di partecipanti volontari. Una volta eseguito l’addestramento, i requisiti computazionali per giocare sono modesti. ALPHASTAR ha vinto partite di StarCraft II in esecuzione su un normale PC desktop con un'unica GPU e anche ALPHAZERO potrebbe essere eseguito su sistemi simili.

5.7 Limitazioni degli algoritmi di ricerca per i giochi

Poiché il calcolo delle decisioni ottime in giochi complessi è intrattabile, tutti gli algoritmi devono effettuare alcune assunzioni e approssimazioni. La ricerca alfa-beta utilizza come approssimazione la funzione di valutazione euristica, mentre la ricerca Monte Carlo calcola una media approssimata su una selezione casuale di simulazioni. La scelta di quale algoritmo usare dipende in parte dalle caratteristiche di ogni gioco: quando il fattore di ramificazione è alto o è difficile definire una funzione di valutazione, si preferisce la ricerca Monte Carlo. Entrambi gli algoritmi, comunque, hanno importanti limitazioni.

Una limitazione della ricerca alfa-beta è la sua vulnerabilità a errori della funzione euristica. La Figura 5.16 mostra un albero di gioco a due strati per cui minimax suggerisce di prendere il ramo di destra poiché $100 > 99$. Quella è la mossa giusta se le valutazioni sono tutte perfettamente accurate, ma supponete che la valutazione di ogni nodo presenti un errore che è indipendente da quello degli altri nodi e che ha una distribuzione casuale con deviazione standard σ . Allora il ramo di sinistra è effettivamente migliore nel 71% dei casi

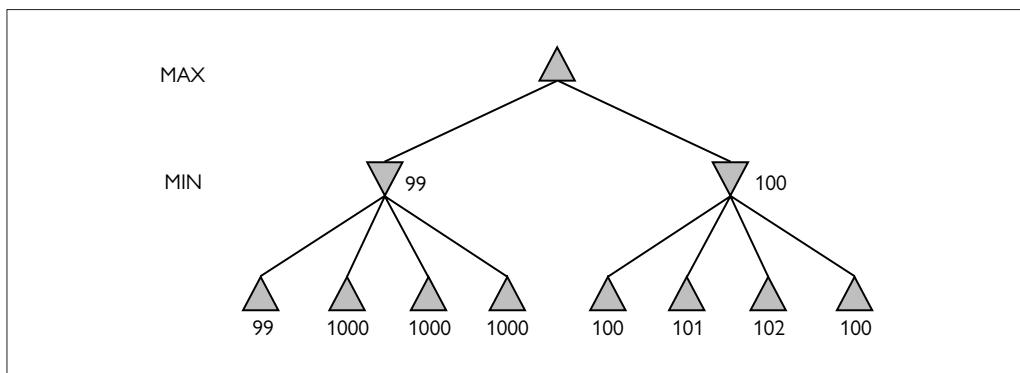


Figura 5.16
Un albero di gioco a due strati per cui minimax euristico potrebbe essere inappropriato.

quando $\sigma = 5$ e nel 58% dei casi quando $\sigma = 2$ (poiché uno dei quattro nodi foglia a destra probabilmente scivolerà sotto 99 in questi casi). Se gli errori nella funzione di valutazione *non* sono indipendenti, le possibilità di commettere un errore aumentano. È difficile compensare questi errori, perché non abbiamo un buon modello delle dipendenze tra i valori di nodi fratelli.

Una seconda limitazione delle ricerche alfa-beta e Monte Carlo è che sono progettate per calcolare (limiti sui) valori di mosse legali. A volte però esiste una mossa che è ovviamente la migliore (per esempio quando esiste una sola mossa legale) e in quel caso non ha senso sprecare tempo di calcolo per calcolare il valore di quella mossa, è meglio eseguirla subito. Un algoritmo di ricerca migliore userebbe il concetto di *utilità dell'espansione di un nodo*, selezionando le espansioni di alta utilità, ovvero quelle che porteranno più probabilmente alla scoperta di una mossa sensibilmente migliore. Se non ci sono più espansioni di nodi la cui utilità è superiore al loro costo (in termini di tempo impiegato), l'algoritmo dovrebbe interrompere la ricerca e fare una mossa. Notate che questo funziona non solo nel caso in cui ci siano mosse chiaramente preferibili ma anche quando le mosse sono *simmetriche*, tali che la ricerca non possa in nessun caso stabilire se una mossa è migliore dell'altra.

Questo tipo di ragionamento riguardante la computazione stessa prende il nome di **metaragionamento** (“ragionamento sul ragionamento”) e non si applica solo ai giochi, ma a ragionamenti di ogni tipo. Tutti i calcoli sono eseguiti al fine di raggiungere decisioni migliori, tutti hanno un costo, e ognuno ha una certa probabilità di avere come risultato un miglioramento della qualità della decisione. La ricerca Monte Carlo tenta di usare il metaragionamento per allocare risorse alle parti più importanti dell’albero, ma non lo fa in modo ottimo.

metaragionamento

Una terza limitazione è che le ricerche alfa-beta e Monte Carlo svolgono tutti i ragionamenti a livello di singole mosse. Gli esseri umani giocano in modo diverso: sono in grado di ragionare a un livello più astratto, considerando un obiettivo di livello superiore – per esempio mettere in trappola la regina dell'avversario – e usandolo per generare in modo *selettivo* piani plausibili. Nel Capitolo 11 studieremo questo tipo di **pianificazione** e nel Paragrafo 11.4 mostreremo come pianificare con una gerarchia di rappresentazioni dalla più astratta alla più concreta.

Un quarto problema è la capacità di incorporare l'**apprendimento automatico** nel processo di ricerca. I primi programmi per i giochi contavano sull'esperienza dell'uomo per funzioni di valutazione, libri sulle aperture, strategie di ricerca, trucchi per migliorare l'efficienza. Stiamo cominciando ora a vedere programmi come ALPHAZERO (Silver *et al.*, 2018) che sfruttano l'apprendimento automatico giocando contro se stessi, anziché affidarsi alle conoscenze specifiche accumulate dall'uomo giocando. Tratteremo in modo approfondito l'apprendimento automatico a partire dal Capitolo 19 del Volume 2.

5.8 Riepilogo

Abbiamo esaminato una varietà di giochi per comprendere che cosa significa giocare in modo ottimo, per comprendere come si possono realizzare in pratica programmi che giocano bene, e per capire come un agente dovrebbe agire in qualsiasi tipo di ambiente con avversari. I concetti più importanti sono i seguenti.

- Un gioco può essere definito specificando lo **stato iniziale** (configurazione del tavoliere all'inizio della partita), le **azioni** legali in ogni stato, il **risultato** di ogni azione, un **test di terminazione** (che determina quando la partita è finita) e una **funzione di utilità** che si applica agli stati terminali per determinare chi ha vinto e qual è il punteggio finale.
- Nei giochi a due giocatori, discreti, deterministicici, a turni, a somma zero e con **informazione perfetta**, l'algoritmo **minimax** può scegliere le mosse ottime eseguendo un'enumerazione in profondità dell'albero di gioco.
- L'algoritmo di ricerca **alfa-beta** calcola le stesse mosse ottime di minimax, ma ha un'efficienza migliore perché elimina sottoalberi che sono sicuramente irrilevanti.
- Normalmente, è impossibile (anche per la ricerca alfa-beta) prendere in considerazione l'intero albero di gioco: di conseguenza, diventa necessario tagliare la ricerca a un certo punto e applicare una **funzione di valutazione** euristica che fornisce la stima dell'utilità di uno stato.
- La **ricerca ad albero Monte Carlo** valuta gli stati non applicando una funzione euristica, ma simulando una partita fino alla fine e usando le regole del gioco per capire chi ha vinto. Poiché le mosse scelte durante la **simulazione** potrebbero non essere quelle ottime, il processo viene ripetuto molte volte e la valutazione viene effettuata calcolando una media dei risultati.
- Molti programmi di giochi precalcolano tabelle delle mosse migliori in apertura e chiusura del gioco in modo da poterle usare per trovare le mosse, anziché effettuare una ricerca pura.
- I giochi che includono elementi casuali possono essere affrontati con **expectiminimax**, un'estensione dell'algoritmo minimax che valuta un **nodo di casualità** prendendo l'utilità media di tutti i figli, pesata in base alla probabilità di ognuno di essi.
- Nei giochi a **informazione imperfetta**, come Kriegspiel e poker, per giocare in modo ottimo bisogna ragionare sugli **stati-credenza** presenti e futuri di ogni giocatore. Un'approssimazione si può ottenere semplicemente facendo la media del valore di ogni azione per ogni possibile configurazione dell'informazione mancante.
- I programmi hanno nettamente superato i campioni umani in giochi come scacchi, dama, Othello, Go, poker e molti altri. Gli esseri umani mantengono un vantaggio in pochi giochi a informazione imperfetta, come bridge e Kriegspiel. In videogiochi come StarCraft e Dota 2, i programmi sono competitivi rispetto ai giocatori umani più forti, ma parte del loro successo potrebbe essere dovuto alla capacità di eseguire molte azioni in modo molto rapido.

Note storiche e bibliografiche

Nel 1846 Charles Babbage discusse la possibilità che i computer potessero giocare a scacchi e dama (Morrison e Morrison, 1961). Non capì la complessità esponenziale degli alberi di ricerca, affermando che: “Le combinazioni del Motore Analitico superano di gran

lunga quelle richieste anche dal gioco degli scacchi”. Babbage inoltre progettò, ma non costruì mai, una macchina speciale per giocare a tris. La prima vera macchina in grado di giocare autonomamente fu costruita intorno al 1890 dall'ingegnere spagnolo Leo-

nardo Torres y Quevedo; era specializzata nel finale scacchistico di re e torre contro re (KRK) e garantiva la vittoria del giocatore con la torre partendo da qualsiasi posizione. L'algoritmo **minimax** si fa risalire a un articolo del 1912 di Ernst Zermelo, il creatore della moderna teoria degli insiemi.

I giochi furono tra le prime attività affrontate nell'IA, con i primi lavori di pionieri come Konrad Zuse (1945), Norbert Wiener nel suo libro *La cibernetica* (1948) e Alan Turing (1953). Ma fu l'articolo di Claude Shannon *Programming a Computer for Playing Chess* (1950) a presentare tutte le idee più importanti: una rappresentazione per le posizioni sulla scacchiera, una funzione di valutazione, la ricerca di quiescenza, e alcune idee per una ricerca selettiva (non esaustiva) dell'albero di gioco. Slater (1950) ebbe l'idea di una funzione di valutazione come combinazione lineare di caratteristiche e sottolineò l'importanza della caratteristica della mobilità negli scacchi.

John McCarthy ideò la ricerca **alfa-beta** nel 1956, anche se solo più tardi fu pubblicato qualcosa al riguardo (Hart ed Edwards, 1961). Knuth e Moore (1975) dimostrarono la correttezza di alfa-beta e ne analizzarono la complessità temporale, mentre Pearl (1982b) mostrò che la ricerca alfa-beta è asintoticamente ottima tra tutti gli algoritmi di ricerca con alberi di gioco di profondità prefissata.

Berliner (1979) introdusse B*, un algoritmo di ricerca euristica che mantiene limiti sull'intervallo dei possibili valori di un nodo nell'albero di gioco anziché assegnare una stima espressa come numero singolo. La ricerca elaborata da David McAllester (1988) e chiamata "conspiracy number search" espande i nodi foglia che, cambiando valore, potrebbero causare un cambiamento della mossa selezionata. MGSS* (Russell e Wefald, 1989) sfrutta le tecniche di teoria delle decisioni presentate nel Capitolo 16 per stimare il valore dell'espansione di ogni foglia in termini di miglioramento atteso nella qualità della decisione alla radice dell'albero.

L'algoritmo SSS* (Stockman, 1979) può essere considerato la versione a due giocatori di A* e non espande mai un numero di nodi maggiore di alfa-beta. I requisiti di memoria lo rendono praticamente inapplicabile, ma ne è stata sviluppata una versione a partire dall'algoritmo RBFS (Korf e Chickering, 1996) che richiede uno spazio lineare. Baum e Smith (1997) proposero di rimpiazzare minimax con una tecnica basata sulla probabilità, dimostrando che per certe categorie di giochi questo porta a risultati migliori. L'algoritmo **expectiminimax** fu proposto da Donald Michie

(1966). Bruce Ballard (1983) estese la potatura alfa-beta per trattare anche gli alberi con nodi di casualità.

Il libro di Pearl *Heuristics* (1984) analizza in modo approfondito molti algoritmi di gioco.

I pionieri della simulazione Monte Carlo furono Metropolis e Ulam (1949) nell'ambito dei calcoli legati allo sviluppo della bomba atomica. La ricerca ad albero Monte Carlo fu introdotta da Ambranson (1987). Tesauro e Galperin (1997) mostrarono come una ricerca Monte Carlo potesse essere combinata con una funzione di valutazione per il gioco del backgammon. La terminazione anticipata della simulazione fu studiata da Lorentz (2015). ALPHAGO era in grado di interrompere le simulazioni e applicare una funzione di valutazione (Silver *et al.*, 2016). Kocsis e Szepesvari (2006) raffinarono l'approccio con il meccanismo di selezione UCT (*upper confidence bounds applied to trees*). Chaslot *et al.* (2008) mostrarono come la ricerca ad albero Monte Carlo potesse essere applicato a un'ampia varietà di giochi e Browne *et al.* (2012) fornirono una panoramica.

Koller e Pfeffer (1997) descrissero un sistema per la risoluzione completa di giochi **parzialmente osservabili**. Tale sistema è in grado di gestire giochi più grandi rispetto ai sistemi precedenti, ma non la versione completa di giochi complessi come poker e bridge. Frank *et al.* (1998) descrissero diverse varianti della ricerca Monte Carlo per giochi parzialmente osservabili, tra cui una in cui MIN ha informazione completa ma MAX no. Schofield e Thielscher (2015) adattarono ai giochi parzialmente osservabili un sistema di gioco generale.

Ferguson elaborò a mano strategie casuali per vincere a Kriegspiel con un alfiere e un cavallo (1992) o due alfieri (1995) contro un re. I primi programmi di Kriegspiel si concentravano sul trovare strategie di scacco matto in chiusura ed eseguivano ricerche AND-OR nello spazio degli stati-credenza (Sakuta e Iida, 2002; Bolognesi e Ciancarini, 2003). Gli algoritmi di stato-credenza incrementali consentirono di trovare strategie di scacco matto a metà partita molto più complesse (Russell e Wolfe, 2005; Wolfe e Russell, 2007), ma la stima efficiente dello stato rimane il principale ostacolo a un gioco generale efficiente (Parker *et al.*, 2005).

Ciancarini e Favini (2010) applicarono la ricerca ad albero Monte Carlo allo Kriegspiel, e Wang *et al.* (2018b) descrissero una versione con stati-credenza della ricerca ad albero Monte Carlo per il Phantom Go.

Le tappe più importanti per il gioco degli **scacchi** sono state segnate dai vincitori del premio Fredkin: BELLE (Condon e Thompson, 1982), il primo pro-

gramma a ottenere il livello di maestro; DEEP THOUGHT (Hsu *et al.*, 1990), il primo a ottenere il livello di maestro internazionale; e Deep Blue (Campbell *et al.*, 2002; Hsu, 2004), che sconfisse il campione del mondo Garry Kasparov in una partita di esibizione nel 1997. Deep Blue eseguiva una ricerca alfa-beta alla velocità di oltre 100 milioni di posizioni al secondo e poteva generare estensioni singole per raggiungere occasionalmente una profondità di 40 strati.

I migliori programmi per gli scacchi di oggi (per esempio STOCKFISH, KOMODO, HOUDINI) superano di gran lunga qualsiasi giocatore umano. Questi programmi hanno ridotto il fattore di ramificazione effettivo a meno di 3 (mentre il fattore di ramificazione reale è di oltre 35), ricercando fino a circa 20 strati a una velocità di circa un milione di nodi al secondo su un computer standard a 1 core. Utilizzano tecniche di potatura come l'euristica della **mossa nulla**, che genera un buon limite inferiore sul valore di una posizione, utilizzando una ricerca poco profonda in cui l'avversario può muovere due volte all'inizio. Inoltre, la **potatura di futilità** aiuta a decidere in anticipo quali mosse causeranno un taglio alfa-beta nei nodi successori. SUNFISH è un programma di scacchi semplificato per scopi didattici; il suo nucleo è composto da meno di 200 righe di codice Python.

L'idea di usare un'analisi retrograda per calcolare tabelle di finali si deve a Bellman (1965). Con questa idea, Ken Thompson (1986, 1996) e Lewis Stiller (1992, 1996) hanno risolto tutti i finali degli scacchi con un massimo di cinque pezzi. Stiller scoprì un solo caso in cui esisteva uno scacco matto forzato, ma richiedeva 262 mosse; questo causò una certa costernazione, perché le regole degli scacchi richiedono che una cattura o una mossa di pedone si verifichi entro 50 mosse, altrimenti si dichiara parità. Nel 2012 Vladimir Makhnychev e Victor Zakharov compilaroni il database di finali Lomonosov Endgame Tablebase, che risolve tutti i finali fino a sette pezzi – alcuni richiedono più di 500 mosse senza una cattura. La tabella per i 7 pezzi occupa 140 terabyte; una tabella per 8 pezzi sarebbe 100 volte più grande.

Nel 2017 ALPHAZERO (Silver *et al.*, 2018) sconfisse STOCKFISH (il campione degli scacchi al computer al campionato 2017 TCEC) in una sfida su 1000 partite, con 155 vittorie e 6 sconfitte. Altre partite portarono a vittorie decisive di ALPHAZERO perfino assegnandogli soltanto un decimo del tempo assegnato a STOCKFISH.

Il gran maestro Larry Kaufman rimase sorpreso dal successo di questo programma basato sulla ricerca Monte Carlo e osservò: “Forse il dominio dei motori

per gli scacchi basati sull'algoritmo minimax sta volgendo al termine, ma è troppo presto per dirlo”. Garry Kasparov commentò: “È un'impresa notevole, anche se dovevamo aspettarcela dopo ALPHAGO. Il programma si avvicina a un approccio di Tipo B simile a quello dell'uomo, come sognavano Claude Shannon e Alan Turing, anziché ricorrere alla forza bruta”. E arrivò poi a predire: “Il gioco degli scacchi è stato scosso alle sue radici da ALPHAZERO, ma questo è solo un piccolo esempio di ciò che verrà. Anche discipline rigide come l'educazione e la medicina saranno scosse” (Sadler e Regan, 2019).

La **dama** fu il primo dei giochi classici a essere giocato dal computer (Strachey, 1952). Arthur Samuel (1952, 1967) sviluppò un programma per giocare a dama capace di apprendere la propria funzione di valutazione giocando contro se stesso, in una forma di apprendimento con rinforzo. È veramente notevole il fatto che Samuel sia stato in grado di creare un programma che giocava meglio di lui su un IBM 704 con sole 10.000 parole di memoria e un processore da 0,000001 GHz. Anche MENACE (*machine educable noughts and crosses engine*, Michie, 1963) utilizzava l'apprendimento con rinforzo per imparare a giocare meglio a tris. Il suo processore era ancora più lento: era costituito da un insieme di 304 scatolette di fiammiferi contenenti perline colorate per rappresentare la migliore mossa appresa in ciascuna posizione.

Nel 1992 il programma per la dama CHINOOK di Jonathan Schaeffer sfidò il leggendario Marion Tinsley, che era stato campione del mondo di dama per più di 20 anni. Tinsley vinse la sfida, ma perse due partite, subendo così la quarta e la quinta sconfitta in tutta la sua carriera. Dopo che Tinsley si dovette ritirare per ragioni di salute, CHINOOK conquistò il trofeo. La saga fu raccontata da Schaeffer (2008).

Nel 2007 Schaeffer e il suo team “risolsero” la dama (Schaeffer *et al.*, 2007): il gioco termina in parità giocando in modo perfetto. Richard Bellman (1965) lo aveva predetto: “Nella dama, il numero di mosse possibili in ogni situazione data è talmente piccolo che possiamo ragionevolmente attenderci una soluzione informatica completa al problema di gioco ottimo”. Bellman tuttavia non aveva previsto l'entità del compito: la tabella dei finali per 10 pezzi contiene 39 trilioni di elementi, e con tale tabella servirono 18 anni-CPU con una ricerca alfa-beta per risolvere il gioco.

I. J. Good, che aveva imparato il gioco del **Go** da Alan Turing, scrisse (1965a): “Penso che programmare un computer per giocare una buona partita a Go sarà ancora più difficile rispetto agli scacchi”. E aveva

ragione: fino al 2015 i programmi per il Go giocavano solo a livello amatoriale. I primi studi sono stati riassunti da Bouzy e Cazenave (2001) e Müller (2002).

Il riconoscimento di pattern visuali fu proposto come tecnica promettente per il Go da Zobrist (1970), mentre Schraudolph *et al.* (1994) analizzarono l'uso dell'apprendimento con rinforzo, Lubberts e Miikkulainen (2001) raccomandarono le reti neurali e Brügmann (1993) introdusse la ricerca ad albero Monte Carlo per il Go. ALPHAGO (Silver *et al.*, 2016) mise insieme queste quattro idee per sconfiggere giocatori professionisti di primo livello come Lee Sedol (con il punteggio di 4–1 nel 2015) e Ke Jie (per 3–0 nel 2016).

Ke Jie osservò: “Dopo che l'umanità ha speso migliaia di anni per migliorare le nostre tattiche, i computer ci dicono che gli esseri umani si sbagliano completamente. Arriverei a dire che nessun singolo essere umano è mai arrivato a toccare la vetta della verità del Go”. Lee Sidol smise di giocare a Go, lamentando: “Anche se diventassi il numero uno, c'è un'entità che non può essere sconfitta”.

Nel 2018, ALPHAZERO sconfisse ALPHAGO al Go, e vinse anche contro i migliori programmi a scacchi e shogi, con un meccanismo di apprendimento attraverso il gioco contro se stesso, senza ricorrere alle conoscenze di esperti umani e senza accedere a partite giocate in passato (naturalmente sono stati esseri umani a definire l'architettura di base, basata su una ricerca ad albero Monte Carlo con reti neurali deep e apprendimento con rinforzo, e a codificare le regole del gioco). Il successo di ALPHAZERO ha fatto aumentare l'interesse per l'apprendimento con rinforzo quale componente chiave dell'IA in generale (cfr. Capitolo 22 del Volume 2). Andando un passo oltre, il sistema MUZERO opera senza nemmeno che gli siano indicate le regole del gioco, deve determinarle da solo giocando. MUZERO ha ottenuto risultati di primo livello giocando a Pacman, scacchi, Go e 75 giochi Atari (Schrittwieser *et al.*, 2019); apprende come generalizzare: per esempio, impara che in Pacman l'azione “su” sposta il giocatore in alto di una casella (a meno che non ci sia un muro), anche se ha osservato il risultato di tale azione soltanto in una piccola percentuale delle posizioni presenti nell'area di gioco.

Othello, chiamato anche Reversi, ha uno spazio di ricerca più piccolo di quello degli scacchi, ma è difficile definire una funzione di valutazione per questo gioco, perché il vantaggio materiale non è importante quanto la mobilità. Esistono programmi per computer di livello superiore all'uomo già dal 1997 (Buro, 2002).

Backgammon, un gioco con casualità, fu analizzato matematicamente da Gerolamo Cardano (1663) e affrontato in informatica con il programma BKG (Berliner, 1980b), che usava una funzione di valutazione costruita a mano e cercava solo a un livello di profondità. BKG fu comunque il primo programma a sconfiggere un campione del mondo umano in un gioco importante (Berliner, 1980a), anche se Berliner riconobbe subito che era stato molto fortunato con i dadi. TD-GAMMON di Gerry Tesauro (1995) imparava la sua funzione di valutazione usando reti neurali addestrate giocando contro se stesso, ed è arrivato a giocare regolarmente a livello dei campioni del mondo, facendo cambiare opinione a molti analisti sulla migliore apertura per diversi lanci di dadi.

Per il **Poker**, come il Go, sono stati compiuti sorprendenti progressi in anni recenti. Bowling *et al.* (2015) hanno usato la teoria dei giochi (cfr. Paragrafo 18.2) per determinare l'esatta strategia ottima per una versione del poker con due giocatori e un numero fisso di rilanci a puntata fissa. Nel 2017, per la prima volta, dei campioni di poker sono stati sconfitti a Texas hold 'em senza limiti a due giocatori, in due sfide separate contro i programmi LIBRATUS (Brown e Sandholm, 2017) e DEEPSTACK (Moravčík *et al.*, 2017). Nel 2019, PLURIBUS (Brown e Sandholm, 2019) ha sconfitto giocatori umani di primo livello in partite di poker Texas hold 'em con sei giocatori. I giochi a più giocatori introducono alcuni problemi di strategia che tratteremo nel Capitolo 18. Petosa e Balch (2019) hanno implementato una versione multiplayer di ALPHAZERO.

Bridge: Smith *et al.* (1998) descrissero il modo in cui BRIDGE BARON vinse il campionato di bridge per computer del 1998 usando piani gerarchici (cfr. Capitolo 11) e azioni di alto livello, come *finessing* e *squeezing*, note ai giocatori di bridge. Ginsberg (2001) descrisse come il suo programma GIB, basato sulla simulazione Monte Carlo (proposta per il bridge da Levy, 1989), vinse il successivo campionato per computer e si comportò sorprendentemente bene contro giocatori umani. Nel XXI secolo i campionati di bridge per computer sono stati dominati da due programmi commerciali, JACK e WBRIDGE5. Nessuno dei due è stato descritto in letteratura, ma si ritiene che entrambi utilizzino tecniche Monte Carlo. In generale, i programmi per il bridge raggiungono il livello dei campioni umani nelle mani effettive, ma restano indietro nella fase di puntata, perché non comprendono del tutto le convenzioni usate dagli umani per comunicare con i loro partner. I programmatori di bridge si sono concentrati maggiormente sulla produzione di

utili programmi didattici che incoraggiassero le persone ad avvicinarsi al gioco, anziché sullo sconfiggere campioni umani.

Scrabble è un gioco in cui i giocatori umani dilettanti hanno difficoltà a trovare parole di alto punteggio, mentre per un computer è facile trovare la parola con il punteggio più alto possibile per una data mano (Gordon, 1994); il difficile sta nel pianificare in anticipo in un gioco parzialmente osservabile e stocastico. Comunque nel 2006 il programma QUACKLE sconfisse l'ex campione del mondo David Boys per 3–2. Boys la prese bene, affermando: “È sempre meglio essere un uomo che un computer”. Una buona descrizione di uno dei migliori programmi, MAVEN, è fornita da Sheppard (2002).

I **videogiochi** come **StarCraft II** implicano centinaia di unità parzialmente osservabili che si muovono in tempo reale con osservazioni quasi continue e multidimensionali⁶ e spazi di azioni con regole complesse. Oriol Vinyals, che è stato campione spagnolo di StarCraft a 15 anni, ha descritto come il gioco possa servire da test e sfida per l'apprendimento con rinforzo (Vinyals *et al.*, 2017a). Nel 2019, Vinyals e il team di DeepMind hanno svelato il programma ALPHASTAR, basato sul deep learning e l'apprendimento con rinforzo, che ha sconfitto giocatori umani esperti 10 partite contro 1 e ha un livello corrispondente allo 0,02% dei migliori giocatori umani con classifica ufficiale (Vi-

nyals *et al.*, 2019). ALPHASTAR ha introdotto un limite al numero di azioni al minuto che può eseguire in situazioni complesse, per reagire alle critiche secondo cui aveva un vantaggio sleale.

I computer hanno sconfitto i migliori giocatori umani in noti videogiochi come Super Smash Bros. (Firoiu *et al.*, 2017), Quake III (Jaderberg *et al.*, 2019) e Dota 2 (Fernandez e Mahlmann, 2018), tutti usando tecniche di deep learning.

I **giochi fisici** come il **calcio robotizzato** (Visser *et al.*, 2008; Barrett e Stone, 2015), il **biliardo** (Lam e Greenspan, 2008; Archibald *et al.*, 2009) e il **ping-pong** (Silva *et al.*, 2015) hanno attratto una certa attenzione nell'IA, poiché uniscono tutte le complicazioni dei videogiochi con la complessità del mondo reale.

Ogni anno si svolgono gare di giochi al computer, come le Computer Olympiad dal 1989. La General Game Competition (Love *et al.*, 2006) si occupa di testare programmi che devono imparare a giocare un gioco ignoto disponendo soltanto di una descrizione logica delle regole. L'International Computer Games Association (ICGA) pubblica l'*ICGA Journal* e gestisce due conferenze biennali, L'International Conference on Computers and Games (ICCG o CG) e l'International Conference on Advances in Computer Games (ACG). L'IEEE pubblica le *IEEE Transactions on Games* e gestisce la Conference on Computational Intelligence and Games, una conferenza annuale.

⁶ A un giocatore umano sembra che gli oggetti si muovano in modo continuo, ma in realtà sono discreti a livello dei pixel dello schermo.

Problemi di soddisfacimento di vincoli

- 6.1 Definizione dei problemi di soddisfacimento di vincoli
- 6.2 Propagazione di vincoli: inferenza nei CSP
- 6.3 Ricerca con backtracking per CSP
- 6.4 Ricerca locale per CSP
- 6.5 La struttura dei problemi
- 6.6 Riepilogo
Note storiche e bibliografiche

In cui vediamo che considerare gli stati qualcosa di più di piccole scatole nere porta a nuovi metodi di ricerca e a una comprensione più profonda della struttura dei problemi.

Nei Capitoli 3 e 4 abbiamo esplorato l'idea che i problemi si possano risolvere eseguendo una ricerca nello spazio degli stati, un grafo in cui i nodi sono stati e gli archi che li collegano sono azioni. Abbiamo visto che euristiche specifiche del dominio del problema consentono di stimare il costo di raggiungere l'obiettivo da un dato stato, ma che dal punto di vista dell'algoritmo di ricerca ogni stato è atomico, o indivisibile: una scatola nera priva di struttura interna. Per ogni problema ci servono informazioni specifiche del dominio per descrivere le transizioni tra gli stati.

In questo capitolo apriamo la scatola nera utilizzando una **rappresentazione fattorizzata** per ogni stato: un insieme di **variabili**, ognuna delle quali ha un **valore**; un problema è risolto quando ogni variabile ha un valore che soddisfa tutti i vincoli su di essa. Un problema descritto in questo modo è detto **problema di soddisfacimento di vincoli**, o **CSP** (*constraint satisfaction problem*).

Gli algoritmi di ricerca CSP sfruttano la struttura degli stati e utilizzano euristiche *di uso generale* anziché *specifiche del dominio* per permettere la soluzione di problemi complessi. L'idea principale è quella di eliminare ampie porzioni dello spazio di ricerca tutte insieme, individuando combinazioni di variabili e valori che violano i vincoli. In più, i CSP hanno il vantaggio che le azioni e il modello di transizione possono essere dedotti dalla descrizione del problema.

6.1 Definizione dei problemi di soddisfacimento di vincoli

Un problema di soddisfacimento di vincoli è costituito da tre componenti \mathcal{X} , \mathcal{D} e \mathcal{C} :

\mathcal{X} è un insieme di variabili, $\{X_1, \dots, X_n\}$.

\mathcal{D} è un insieme di domini, $\{D_1, \dots, D_n\}$, uno per ogni variabile.

\mathcal{C} è un insieme di vincoli che specificano combinazioni di valori ammesse.

relazione

assegnamento
consistente
completo
soluzione
assegnamento
parziale
soluzione parziale

Un dominio D_i è costituito da un insieme di valori ammessi, $\{v_1, \dots, v_k\}$, per la variabile X_i . Per esempio, una variabile booleana avrebbe il dominio $\{\text{vero}, \text{falso}\}$. Variabili diverse possono avere domini differenti e di dimensioni diverse. Ogni vincolo C_i è costituito da una coppia $\langle \text{ambito}, \text{rel} \rangle$, dove **ambito** è una tupla di variabili che partecipano nel vincolo e **rel** è una **relazione** che definisce i valori che tali variabili possono assumere. Una relazione può essere rappresentata come insieme esplicito di tutte le tuple di valori che soddisfano il vincolo, o come una funzione che è in grado di determinare se una tupla è membro della relazione. Per esempio, Se X_1 e X_2 hanno entrambe il dominio $\{1, 2, 3\}$, allora il vincolo che afferma che X_1 deve essere maggiore di X_2 si può scrivere come $\langle(X_1, X_2), \{(3, 1), (3, 2), (2, 1)\}\rangle$ o come $\langle(X_1, X_2), X_1 > X_2\rangle$.

Nei CSP si effettuano **assegnamenti** di valori a variabili, $\{X_i = v_i, X_j = v_j, \dots\}$. Un assegnamento che non viola alcun vincolo è chiamato **consistente** o legale. Un assegnamento è **completo** se a tutte le variabili è assegnato un valore; una **soluzione** di un problema CSP è un assegnamento completo e consistente. Un assegnamento è **parziale** se alcune delle variabili rimangono non assegnate, e una **soluzione parziale** è un assegnamento parziale e consistente. Risolvere un CSP è un problema NP-completo in generale, anche se esistono importanti sottoclassi di CSP che si possono risolvere in modo molto efficiente.

6.1.1 Un problema di esempio: colorazione di una mappa

Supponiamo di esserci stanchi della Romania e di considerare una mappa dell'Australia che mostra tutti i suoi stati e territori, come si vede nella Figura 6.1(a). Abbiamo il compito di co-

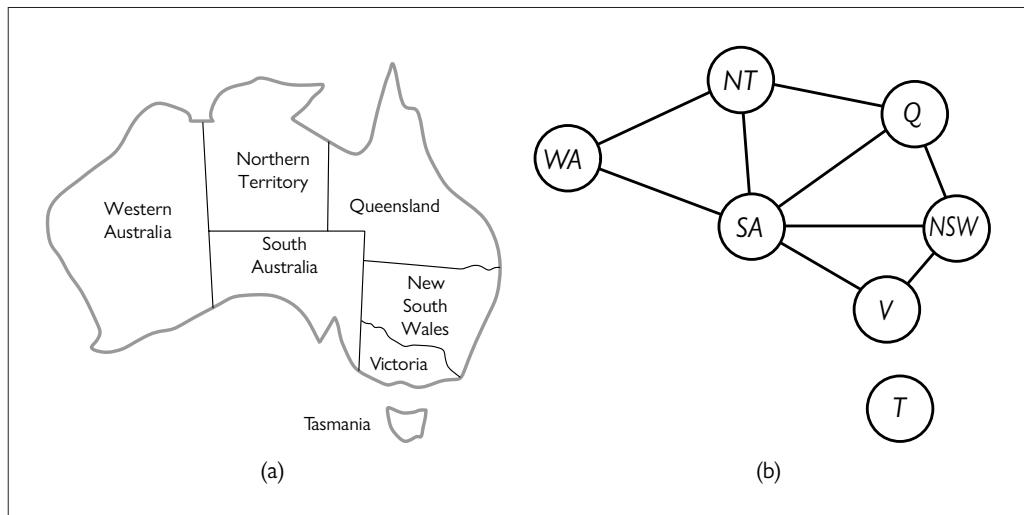


Figura 6.1 (a) I principali stati e territori dell'Australia. La colorazione di questa mappa può essere vista come un problema di soddisfacimento di vincoli (CSP). L'obiettivo è assegnare un colore a ogni regione in modo che non esistano due regioni adiacenti con lo stesso colore. (b) Il problema di colorazione della mappa dell'Australia rappresentato sotto forma di grafo di vincoli.

lorare ogni regione di rosso, verde o blu in modo tale che non esistano regioni adiacenti dello stesso colore. Per formulare questo come un CSP, definiamo una variabile per ogni regione:

$$\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}.$$

Il dominio di ogni variabile è l'insieme $D_i = \{\text{rosso, verde, blu}\}$. I vincoli richiedono che le regioni adiacenti abbiano colori distinti. Poiché ci sono confini tra le regioni, ci sono nove vincoli:

$$\begin{aligned} \mathcal{C} = \{ &SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ &WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V \}. \end{aligned}$$

Stiamo utilizzando delle forme abbreviate; $SA \neq WA$ è una forma abbreviata per $\{(SA, WA), (SA \neq WA)\}$, dove $SA \neq WA$ a sua volta può essere enumerato interamente come:

$$\{(rosso, verde), (rosso, blu), (verde, rosso), (verde, blu), (blu, rosso), (blu, verde)\}.$$

Ci sono molte soluzioni possibili per questo problema, come:

$$\{WA = \text{rosso}, NT = \text{verde}, Q = \text{rosso}, NSW = \text{verde}, V = \text{rosso}, SA = \text{blu}, T = \text{rosso}\}.$$

Può essere utile visualizzare un CSP come un **grafo di vincoli** (Figura 6.1(b)). I nodi del grafo corrispondono alle variabili del problema e un arco connette ogni coppia di variabili che partecipano a un vincolo.

grafo di vincoli

Perché esprimere un problema sotto forma di CSP? Un motivo è che i CSP costituiscono una rappresentazione naturale per un'ampia varietà di problemi; spesso è facile formulare un problema come CSP. Un altro motivo è che sono stati spesi anni di lavoro per sviluppare risolutori CSP veloci ed efficienti. Un terzo motivo è che un risolutore CSP può potare rapidamente grandi porzioni dello spazio degli stati, cosa che un algoritmo di ricerca in uno spazio degli stati atomico non potrebbe fare. Per esempio, una volta che abbiamo scelto $\{SA = \text{blu}\}$ nel problema dell'Australia, possiamo concludere che nessuna delle cinque variabili adiacenti può assumere il valore *blu*. Una procedura di ricerca che non usi vincoli dovrebbe considerare $3^5 = 243$ assegnamenti per le cinque variabili adiacenti; con i vincoli, invece, abbiamo soltanto $2^5 = 32$ assegnamenti da considerare, con una riduzione dell'87%.

Nel caso di una ricerca in uno spazio degli stati atomico possiamo soltanto chiederci se uno specifico stato è un obiettivo, e se non lo è passare a un altro. Nei problemi CSP, una volta determinato che un assegnamento parziale viola un vincolo, possiamo immediatamente scartare ulteriori raffinamenti di tale assegnamento. Inoltre, possiamo vedere perché l'assegnamento non è una soluzione (vediamo quali variabili violano un vincolo) e quindi focalizzare la nostra attenzione sulle variabili che contano. Di conseguenza, molti problemi intrattabili per la ricerca su spazio degli stati atomico possono essere risolti rapidamente quando sono formulati come CSP.

6.1.2 Un problema di esempio: programmazione di lavori

Nelle fabbriche c'è il problema di programmare i vari lavori che rientrano nelle attività quotidiane, soggetti a vari vincoli (il cosiddetto *job-shop scheduling*). Nella pratica, molti di questi problemi sono risolti utilizzando tecniche CSP. Consideriamo il problema di programmare l'assemblaggio di un'automobile. L'intero lavoro è composto da vari compiti, ognuno dei quali può essere modellato da una variabile il cui valore è il tempo al quale il compito inizia, espresso come numero intero di minuti. I vincoli possono stabilire che un compito deve essere svolto prima di un altro: per esempio, l'installazione di una ruota deve avvenire prima dell'installazione del copriruota, che solo un certo numero massimo di compiti possono essere svolti contemporaneamente, e anche che un compito richiede una certa quantità di tempo per essere portato a termine.

Consideriamo una piccola parte dell’assemblaggio di un’auto, costituita da 15 compiti: installare gli assi (anteriore e posteriore), fissare tutte e quattro le ruote (destra e sinistra, anteriore e posteriore), stringere i dadi per ogni ruota, fissare i copriruota e ispezionare l’assemblaggio finale. Possiamo rappresentare questi compiti con 15 variabili:

$$\mathcal{X} = \{Asse_A, Asse_P, Ruota_{DA}, Ruota_{SA}, Ruota_{DP}, Ruota_{SP}, Dadi_{DA}, \\ Dadi_{SA}, Dadi_{DP}, Dadi_{SP}, Copri_{DA}, Copri_{SA}, Copri_{DP}, Copri_{SP}, Ispezione\}.$$

vincolo di precedenza

Ora rappresentiamo i **vincoli di precedenza** tra singoli compiti. Ogni volta che un compito T_1 deve essere svolto prima del compito T_2 , e il compito T_1 richiede la durata d_1 per essere portato a termine, aggiungiamo un vincolo aritmetico della forma:

$$T_1 + d_1 \leq T_2.$$

Nel nostro esempio, gli assi devono essere già al loro posto prima che le ruote siano fissate, e l’installazione di un asse richiede 10 minuti, perciò scriviamo:

$$Asse_A + 10 \leq Ruota_{DA}; Asse_A + 10 \leq Ruota_{SA}; \\ Asse_P + 10 \leq Ruota_{DP}; Asse_P + 10 \leq Ruota_{SP}.$$

Ora diciamo che, per ogni ruota, dobbiamo fissare la ruota stessa (compito che richiede 1 minuto), poi stringere i dadi (2 minuti) e infine installare il copriruota (1 minuto, ma non è ancora rappresentato):

$$\begin{array}{ll} Ruota_{DA} + 1 \leq Dadi_{DA}; & Dadi_{DA} + 2 \leq Copri_{DA}; \\ Ruota_{SA} + 1 \leq Dadi_{SA}; & Dadi_{SA} + 2 \leq Copri_{SA}; \\ Ruota_{DP} + 1 \leq Dadi_{DP}; & Dadi_{DP} + 2 \leq Copri_{DP}; \\ Ruota_{SP} + 1 \leq Dadi_{SP}; & Dadi_{SP} + 2 \leq Copri_{SP}. \end{array}$$

vincolo disgiuntivo

Supponiamo di avere quattro lavoratori per installare ruote, che però devono condividere un solo strumento per facilitare il posizionamento dell’asse. Ci serve un **vincolo disgiuntivo** che affermi che $Asse_A$ e $Asse_P$ non devono sovrapporsi nel tempo; uno dei due deve venire prima dell’altro:

$$(Asse_A + 10 \leq Asse_P) \text{ or } (Asse_P + 10 \leq Asse_A).$$

Questo sembra un vincolo più complesso, che combina aritmetica e logica, ma alla fine si riduce a un insieme di coppie di valori che $Asse_A$ e $Asse_P$ possono assumere.

Dobbiamo anche stabilire che l’ispezione arriva per ultima e richiede 3 minuti. Per ogni variabile eccetto *Ispezione* aggiungiamo un vincolo della forma $X + d_X \leq Ispezione$. Infine, supponiamo che vi sia un requisito per cui l’intero assemblaggio debba essere svolto in 30 minuti. Possiamo esprimere questo limitando il dominio di tutte le variabili:

$$D_i = \{1, 2, 3, \dots, 30\}.$$

Questo particolare problema si risolve in modo banale, ma i CSP sono stati applicati con successo a problemi di programmazione (*scheduling*) di lavori come questo ma con migliaia di variabili.

6.1.3 Varianti del formalismo CSP

dominio discreto dominio finito

Nella forma più semplice di CSP le variabili hanno domini **discreti** e **finiti**. I problemi di colorazione di mappe e programmazione di lavori con limiti temporali rientrano in questa categoria. Anche il problema delle 8 regine (Figura 4.3) può essere considerato un CSP a dominio finito, in cui le variabili Q_1, \dots, Q_8 corrispondono alle posizioni di ogni regina nelle colonne da 1 a 8 e il dominio di ogni variabile specifica i possibili numeri di riga per la regina in quella colonna, $D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$. I vincoli stabiliscono che non possono esserci due regine nella stessa riga o diagonale.

Un dominio discreto può essere **infinito**, come l'insieme degli interi o delle stringhe (se non avessimo posto una scadenza temporale nel problema della programmazione di lavori, ci sarebbe stato un numero infinito di tempi di inizio per ogni variabile). Con domini infiniti, dobbiamo usare vincoli impliciti come $T_1 + d_1 \leq T_2$ anziché tuple esplicite di valori. Esistono algoritmi speciali (che non discuteremo qui) per trattare **vincoli lineari** su variabili intere, come quello che abbiamo appena scritto: in questi vincoli ogni variabile appare solo in forma lineare. È possibile mostrare che non esiste alcun algoritmo per risolvere problemi generali con **vincoli non lineari** su variabili intere: il problema è indecidibile.

I problemi di soddisfacimento di vincoli con **domini continui** sono comuni nel mondo reale, e sono stati studiati approfonditamente nel campo della ricerca operativa. Compilare il programma degli esperimenti per il telescopio spaziale Hubble, per esempio, richiede una temporizzazione estremamente precisa delle osservazioni; l'inizio e la fine di ognuna di esse deve rispettare una serie di vincoli astronomici, di precedenza e di gestione energetica. La categoria meglio conosciuta di CSP a dominio continuo è quella dei problemi di **programmazione lineare**, in cui i vincoli devono essere espressi come uguaglianze o disuguaglianze lineari. I problemi di programmazione lineare possono essere risolti in un tempo polinomiale nel numero delle variabili. Sono stati studiati anche problemi con tipi diversi di vincoli e di funzioni obiettivo: programmazione quadratica, programmazione conica del secondo ordine e così via. Questi problemi costituiscono un importante campo di studio per la matematica applicata.

Oltre a esaminare i tipi di variabili che possono apparire in un CSP, è utile considerare anche i tipi di vincolo. Quello più semplice è il **vincolo unario**, che interessa il valore di una singola variabile. Per esempio, nel problema di colorazione di una mappa potrebbe darsi che gli australiani del sud detestino il colore verde; potremmo esprimere questo fatto con il vincolo unario $\langle (SA), SA \neq \text{verde} \rangle$. Anche la specificazione iniziale del dominio di una variabile può essere vista come un vincolo unario.

Un **vincolo binario**, come $SA \neq NSW$, mette in relazione due variabili. Un CSP si dice **binario** quando comprende solo vincoli unari e binari; come si vede nella Figura 6.1(b) può essere rappresentato con un grafo dei vincoli.

Possiamo anche definire vincoli di ordine più alto. Per esempio, il vincolo ternario $Fra(X, Y, Z)$ può essere definito come $\langle (X, Y, Z), X < Y < Z \text{ or } X > Y > Z \rangle$.

Un vincolo che interessa un numero arbitrario di variabili si dice **vincolo globale** (il termine è utilizzato comunemente ma rischia di generare confusione, perché tale vincolo non interessa necessariamente *tutte* le variabili del problema). Uno dei più comuni vincoli globali è *Tuttediverse*, che afferma che tutte le variabili interessate dal vincolo devono avere valori diversi. Nei problemi di Sudoku (cfr. Paragrafo 6.2.6), tutte le variabili di una riga, colonna o riquadro 3×3 devono soddisfare un vincolo *Tuttediverse*. Un altro esempio sono i giochi enigmistici di **criptoaritmetica** (Figura 6.2(a)). In questi giochi ogni lettera corrisponde a una cifra diversa. Nel caso della Figura 6.2(a), si potrebbe scrivere un vincolo a sei variabili $Tuttediverse(F, T, U, W, R, O)$. I vincoli imposti dall'aritmetica dell'addizione sulle quattro colonne del rompicapo possono essere scritti come i seguenti vincoli n -ari:

$$\begin{aligned} O + O &= R + 10 \cdot C_1 \\ C_1 + W + W &= U + 10 \cdot C_2 \\ C_2 + T + T &= O + 10 \cdot C_3 \\ C_3 = F, \end{aligned}$$

dove C_1, C_2 e C_3 sono variabili ausiliarie che rappresentano la cifra del riporto nella colonna delle decine, centinaia o migliaia. Questi vincoli possono essere rappresentati in un **ipergrafo di vincoli**, come quello mostrato nella Figura 6.2(b). Un ipergrafo è costituito da nodi normali (i cerchi nella figura) e ipernodi (i quadrati) che rappresentano vincoli n -ari, cioè con n variabili.

dominio infinito

vincolo lineare

vincolo non lineare

dominio continuo

vincolo unario

CSP binario

vincolo globale

criptoaritmetica

ipergrafo di vincoli

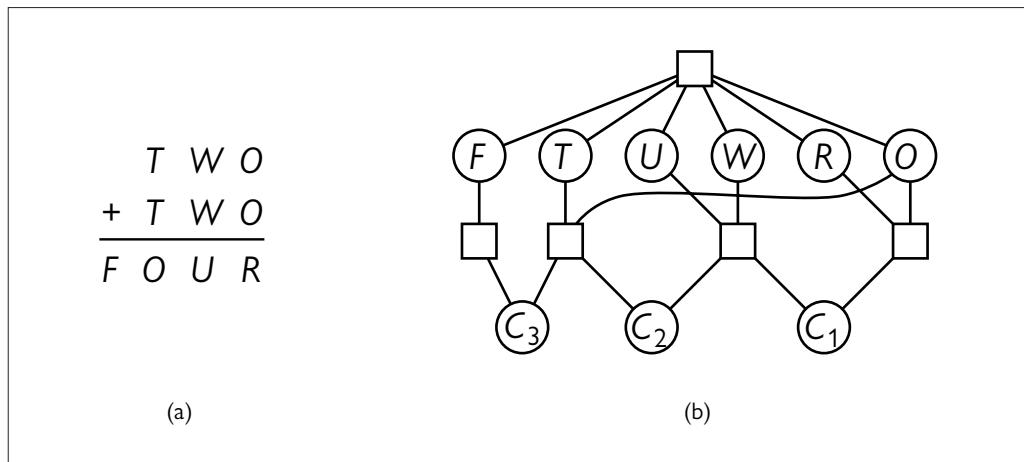


Figura 6.2 (a) Un problema di criptoaritmetica. Ogni lettera indica una singola cifra distinta; lo scopo è trovare una corrispondenza tra lettere e cifre tale che la somma risulti aritmeticamente corretta, con il vincolo aggiuntivo che non sono permessi zeri aggiuntivi alla sinistra dei numeri. (b) L'ipergrafo dei vincoli per il problema di criptoaritmetica, che mostra sia il vincolo *Tuttediverse* (il nodo quadrato in alto) che quelli sull'addizione tra le colonne (i quattro nodi quadrati al centro). Le variabili C_1 , C_2 e C_3 rappresentano le cifre di riporto per le tre colonne.

grafo duale

In alternativa, ogni vincolo a dominio finito può essere ridotto a un insieme di vincoli binari se si introduce un numero sufficiente di variabili ausiliarie (nell'Esercizio 6.NARY viene chiesto di fornire la dimostrazione). Questo significa che potremmo trasformare qualsiasi CSP in un problema con soli vincoli binari, cosa che renderebbe certamente più semplice il compito di chi progetta gli algoritmi. Un altro modo per convertire un CSP n -ario in uno binario è la trasformazione del **grafo duale**: si crea un nuovo grafo in cui vi sarà una variabile per ogni vincolo del grafo originale e un vincolo binario per ogni coppia di vincoli del grafo originale che condividono le variabili.

Per esempio, consideriamo un CSP con le variabili $\mathcal{X} = \{X, Y, Z\}$, ognuna con il dominio $\{1, 2, 3, 4, 5\}$, e con i due vincoli $C_1 : \langle(X, Y, Z), X + Y = Z\rangle$ e $C_2 : \langle(X, Y), X + 1 = Y\rangle$. Allora il grafo duale conterebbe le variabili $\mathcal{X} = \{C_1, C_2\}$, dove il dominio della variabile C_1 nel grafo duale è l'insieme delle tuple $\{(x_i, y_j, z_k)\}$ dal vincolo C_1 del problema originale, e in modo simile il dominio di C_2 è l'insieme delle tuple $\{(x_i, y_j)\}$. Il grafo duale ha il vincolo binario $\langle(C_1, C_2), R_1\rangle$, dove R_1 è una nuova relazione che definisce il vincolo tra C_1 e C_2 ; in questo caso sarebbe $R_1 = \{\langle(1, 2, 3), (1, 2)\rangle, \langle(2, 3, 5), (2, 3)\rangle\}$.

Esistono tuttavia due motivi per cui potremmo preferire un vincolo globale come *Tuttediverse* a un insieme di vincoli binari. Per prima cosa, formalizzare un problema con *Tuttediverse* risulta più facile e presenta meno rischi di commettere errori. In secondo luogo, per vincoli globali è possibile progettare specifici algoritmi inferenziali che sono più efficienti rispetto all'uso di vincoli primitivi. Descriveremo questi algoritmi inferenziali nel Paragrafo 6.2.5.

vincolo di preferenza

I vincoli che abbiamo descritto fin qui sono tutti assoluti; la loro violazione impedisce di trovare una soluzione. Molti CSP nel mondo reale includono **vincoli di preferenza** che indicano quali soluzioni sono appunto preferibili. Per esempio, nel problema della generazione dell'orario per un corso di studi, ci sono vincoli assoluti per cui nessun professore può tenere due lezioni nello stesso tempo, ma potremmo anche consentire dei vincoli di preferenza: il Prof. R potrebbe gradire di lavorare il mattino, il Prof. N invece il pomeriggio. Un orario che prevedesse lezioni del Prof. R alle 14:00 sarebbe ancora una soluzione ammessa (a meno che il Prof. R per caso non sia anche il preside), ma non ottima. Spesso i vincoli di preferenza

possono essere rappresentati come costi sugli assegnamenti di singole variabili: per esempio, assegnare al Prof. R delle lezioni al pomeriggio costerà 2 punti nella funzione obiettivo globale, mentre le lezioni al mattino costeranno solo 1. Con questa formulazione, i CSP con vincoli di preferenza possono essere risolti usando metodi di ricerca per l'ottimizzazione, locali o basati sul cammino. Parliamo in questo caso di un **problema di ottimizzazione di vincoli**, o COP (*constraint optimization problem*). I problemi di programmazione lineare sono una classe di COP.

**problema
di ottimizzazione
di vincoli**

6.2 Propagazione di vincoli: inferenza nei CSP

Un algoritmo di ricerca in uno spazio degli stati atomico procede in un solo modo: espandendo un nodo per visitare i successori. Un algoritmo CSP ha delle scelte: può generare successori scegliendo un nuovo assegnamento di variabile, oppure può effettuare un tipo specifico di inferenza denominato **propagazione dei vincoli**, cioè utilizzando i vincoli per ridurre il numero di valori legali per una variabile, il che a sua volta può ridurre i valori legali per un'altra variabile e così via. L'idea è che in questo modo rimarranno meno scelte da considerare quando occorrerà effettuare la prossima scelta di un assegnamento di variabile. La propagazione dei vincoli può essere intrecciata con la ricerca, oppure può essere svolta come passo di elaborazione preliminare, prima che la ricerca cominci. Talvolta questa elaborazione preliminare può risolvere l'intero problema, per cui non è richiesta alcuna ricerca.

**propagazione
dei vincoli**

Il concetto fondamentale è la **consistenza locale**. Se consideriamo ogni variabile come un nodo in un grafo (Figura 6.1(b)) e ogni vincolo binario come un arco, il processo di forzare la consistenza locale in ogni parte del grafo causa l'eliminazione dei valori inconsistenti in tutto il grafo. Esistono diversi tipi di consistenza locale, che trattiamo nel seguito uno per uno.

consistenza locale

6.2.1 Consistenza di nodo

Una singola variabile (corrispondente a un nodo nel grafo CSP) è **nodo-consistente** se tutti i valori del suo dominio soddisfano i suoi vincoli unari. Per esempio, nella variante del problema di colorazione della mappa dell'Australia (Figura 6.1), dove gli australiani del sud non amano il verde, la variabile *SA* inizia con il dominio {rosso, verde, blu} e possiamo renderla nodo-consistente eliminando il *verde*, lasciandole quindi il dominio ridotto {rosso, blu}. Diciamo che un grafo è nodo-consistente se ogni variabile del grafo è nodo-consistente.

consistenza di nodo

È facile eliminare tutti i vincoli unari in un CSP riducendo il dominio delle variabili con vincoli unari all'inizio del processo di risoluzione. Come si è detto in precedenza, è anche possibile trasformare tutti i vincoli *n*-ari in binari. Per questo motivo, alcuni risolutori di CSP operano solo con vincoli binari, presupponendo che l'utente elimini prima gli altri vincoli. Facciamo nostra questa ipotesi per la parte restante di questo capitolo, ove non sia indicato diversamente.

6.2.2 Consistenza d'arco

Una variabile in un CSP si dice **arco-consistente**¹ se ogni valore del suo dominio soddisfa i suoi vincoli binari. In termini più formali, X_i è arco-consistente rispetto a un'altra variabile X_j se per ogni valore nel dominio corrente D_i c'è un valore nel dominio D_j che soddisfa il vincolo binario sull'arco (X_i, X_j) . Un grafo è arco-consistente se ogni variabile è arco-consi-

consistenza d'arco

¹ Nella terminologia dei grafi si parla di archi o anche di lati, per cui potrebbe avere senso usare il termine latto-consistente, ma arco-consistente è decisamente più diffuso.

```

function AC-3(csp) returns false se viene trovata una inconsistenza, o altrimenti true
  coda  $\leftarrow$  una coda di archi, inizialmente tutti quelli in csp

  while coda non è vuota do
     $(X_i, X_j) \leftarrow \text{POP}(\textit{coda})$ 
    if RIMUOVI-VALORI-INCONSISTENTI(csp,  $X_i$ ,  $X_j$ ) then
      if dimensione di  $D_i = 0$  then return false
      for each  $X_k$  in  $X_i$ -ADIACENTI – { $X_j$ } do
        aggiungi  $(X_k, X_i)$  a coda
    return true

function RIMUOVI-VALORI-INCONSISTENTI(csp,  $X_i$ ,  $X_j$ ) returns true se e solo se viene rimosso un valore
  rimosso  $\leftarrow$  false
  for each  $x$  in  $D_i$  do
    if nessun valore  $y$  in  $D_j$  permette a  $(x, y)$  di soddisfare il vincolo tra  $X_i$  e  $X_j$  then
      rimuovi  $x$  da  $D_i$ 
      rimosso  $\leftarrow$  true
  return rimosso

```

Figura 6.3 L'algoritmo AC-3 per il controllo della consistenza d'arco. Dopo l'applicazione di AC-3 ogni arco è arco-consistente, oppure qualche variabile avrà un dominio vuoto, il che significa che il CSP non può essere risolto. Il nome "AC-3" fu usato dall'inventore dell'algoritmo (Mackworth, 1977) perché era la terza versione presentata all'interno dell'articolo.

stente con ogni altra. Per esempio, consideriamo il vincolo $Y = X^2$ dove il dominio di X e di Y è l'insieme delle cifre decimali. Possiamo scrivere esplicitamente tale vincolo come:

$$\langle(X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\}\rangle.$$

Per rendere X arco-consistente rispetto a Y , riduciamo il dominio di X a $\{0, 1, 2, 3\}$. Se rendiamo anche Y arco-consistente rispetto a X , allora il dominio di Y diventa $\{0, 1, 4, 9\}$ e l'intero CSP è arco-consistente.

D'altra parte, la consistenza d'arco non serve a nulla nel problema della colorazione della mappa dell'Australia. Consideriamo il seguente vincolo di diseguaglianza su (SA, WA) :

$$\{(rosso, verde), (rosso, blu), (verde, rosso), (verde, blu), (blu, rosso), (blu, verde)\}.$$

A prescindere dal valore scelto per SA (o per WA), esiste un valore valido per l'altra variabile. Perciò l'applicazione della consistenza d'arco non ha effetto sui domini delle variabili.

Il più noto algoritmo per controllare consistenza d'arco si chiama AC-3 (Figura 6.3). Per rendere ogni variabile arco-consistente, tale algoritmo mantiene una coda di archi da considerare (in effetti l'ordine di considerazione non conta, perciò la struttura dati in realtà è un insieme, ma per tradizione si parla di coda). Inizialmente la coda contiene tutti gli archi del CSP. AC-3 poi estrae un arco arbitrario (X_i, X_j) dalla coda e rende X_i arco-consistente rispetto a X_j . Se questo lascia invariato D_i , l'algoritmo passa all'arco successivo; se invece D_i cambia (riducendosi), aggiungiamo alla coda tutti gli archi (X_k, X_i) , dove X_k è adiacente a (detto anche vicino di) X_i . Dobbiamo fare ciò perché il cambiamento in D_i potrebbe consentire ulteriori riduzioni in D_k , anche se abbiamo già considerato precedentemente X_k . Se D_i è ridotto all'insieme vuoto, sappiamo che l'intero problema CSP non ha una soluzione consistente e AC-3 può restituire subito il fallimento. Altrimenti continuiamo a controllare,

cercando di rimuovere valori dai domini delle variabili finché non vi sono più archi nella coda. A quel punto ci rimane un CSP che è equivalente all'originale (entrambi hanno le stesse soluzioni), ma nella maggior parte dei casi la ricerca nella versione arco-consistente sarà più rapida, perché le variabili hanno domini più piccoli. In alcuni casi il problema si risolve completamente (riducendo ogni dominio alla dimensione 1) e in altri si dimostra che non esiste soluzione (riducendo alcuni domini alla dimensione 0).

La complessità di AC-3 può essere analizzata come segue. Supponiamo di avere un CSP con n variabili, ognuna con dimensione del dominio non superiore a d , e con c vincoli binari (archi). Ogni arco (X_k, X_i) può essere inserito nella coda soltanto d volte perché X_i ha al più d valori che si possono eliminare. Il controllo della consistenza di un arco può essere svolto in un tempo $O(d^2)$, perciò otteniamo un tempo totale del caso peggiore $O(cd^3)$.

6.2.3 Consistenza di cammino

Supponiamo di voler colorare la mappa dell'Australia, ma con due soli colori ammessi: rosso e blu. La consistenza d'arco non serve a nulla, perché ogni variabile è già arco-consistente: ognuna può essere rossa con il blu all'altro estremo dell'arco, o vice versa. Ma qui è chiaro che non c'è soluzione al problema: poiché Western Australia, Northern Territory e South Australia confinano tutte tra loro, ci servono almeno tre colori solo per queste.

La consistenza d'arco restringe i domini (vincoli unari) utilizzando gli archi (vincoli binari). Per fare progressi in problemi quali la colorazione delle mappe ci serve una nozione di consistenza più forte. La **consistenza di cammino** restringe i vincoli binari utilizzando vincoli impliciti che sono inferiti considerando triplete di variabili.

consistenza
di cammino

Un insieme di due variabili $\{X_i, X_j\}$ è cammino-consistente rispetto a una terza variabile X_m se, per ogni assegnamento $\{X_i = a, X_j = b\}$ consistente con i vincoli (se esistono) su $\{X_i, X_j\}$, esiste un assegnamento di X_m che soddisfa i vincoli su $\{X_i, X_m\}$ e $\{X_m, X_j\}$. Il termine consistenza di cammino si riferisce alla consistenza complessiva del cammino da X_i a X_j con X_m nel mezzo.

Vediamo in che modo la consistenza di cammino può risultare utile nella colorazione della mappa dell'Australia con due soli colori. Renderemo l'insieme $\{WA, SA\}$ cammino-consistente rispetto a NT . Iniziamo enumerando gli assegnamenti consistenti all'insieme. In questo caso ce ne sono solo due: $\{WA = \text{rosso}, SA = \text{blu}\}$ e $\{WA = \text{blu}, SA = \text{rosso}\}$. Vediamo che con entrambi questi assegnamenti NT non può essere né *rosso* né *blu* (perché entrerebbe in conflitto con WA o SA). Poiché non esiste una scelta valida per NT , eliminiamo entrambi gli assegnamenti e ci ritroviamo senza alcun assegnamento valido per $\{WA, SA\}$. Perciò sappiamo che non può esistere soluzione per questo problema.

6.2.4 K-consistenza

Forme più potenti di propagazione possono essere definite usando il concetto di **k-consistenza**. Un CSP è k -consistente se, per ogni insieme di $k-1$ variabili e ogni loro assegnamento consistente, è sempre possibile assegnare un valore consistente a ogni k -esima variabile. La 1-consistenza significa che, dato l'insieme vuoto, possiamo rendere consistente ogni insieme di una sola variabile; è il caso della consistenza di nodo. La 2-consistenza corrisponde alla consistenza d'arco. Per grafi con vincoli binari, la 3-consistenza corrisponde alla consistenza di cammino.

k-consistenza

Un CSP è **fortemente k-consistente** se è k -consistente e anche $(k-1)$ -consistente, $(k-2)$ -consistente, . . . fino a 1-consistente. Ora supponiamo di avere un CSP con n nodi e di renderlo fortemente n -consistente (come dire, fortemente k -consistente con $k = n$). Allora è possibile risolvere quel problema come segue: per prima cosa scegliamo un valore consistente per X_1 . Dato che il grafo è 2-consistente saremo certamente in grado di scegliere un valore per X_2 , ma anche per X_3 vale lo stesso discorso perché il grafo è 3-consistente, e così

fortemente
k-consistente

via: per ogni variabile X_i dobbiamo solo cercare nei d valori del dominio per trovare un valore consistente con X_1, \dots, X_{i-1} . Il tempo di esecuzione totale è soltanto $O(n^2d)$.

Naturalmente, non si può ottenere un simile risultato gratis: il problema del soddisfacimento di vincoli è NP-completo in generale, e ogni algoritmo che stabilisce la n -consistenza, nel caso pessimo, avrà una complessità esponenziale in n . Cosa ancora peggiore, la n -consistenza richiede anche uno spazio esponenziale in n . Il problema della memoria è ancora più grave di quello del tempo. Nella pratica, determinare il livello di consistenza appropriato è per lo più un procedimento empirico. Il calcolo della 2-consistenza viene effettuato comunemente, quello della 3-consistenza più raramente.

6.2.5 Vincoli globali

Ricordate che un **vincolo globale** è un vincolo che interessa un numero arbitrario di variabili (ma non necessariamente tutte). I vincoli globali si presentano frequentemente nei problemi reali e possono essere gestiti con algoritmi speciali più efficienti dei metodi generici descritti fin qui. Per esempio, il vincolo *Tuttediverse* richiede che le variabili specificate abbiano tutti valori diversi (come nel precedente problema di criptoaritmetica e nei problemi di Sudoku riportati più avanti). Per il vincolo *Tuttediverse* un semplice metodo per rilevare le inconsistenze può essere il seguente: se il vincolo coinvolge m variabili, e se queste hanno complessivamente n possibili valori distinti, e $m > n$, allora il vincolo non può essere soddisfatto.

Questa considerazione porta al semplice algoritmo: si rimuove prima di tutto ogni variabile coinvolta nel vincolo che ha un dominio con un solo valore, cancellando tale valore dai domini di tutte le altre variabili. Si ripete finché ci sono variabili con domini a un solo valore: se in qualsiasi momento un dominio rimane vuoto o ci sono più variabili che valori rimasti, è stata scoperta un'inconsistenza.

Possiamo applicare questo metodo per rilevare l'inconsistenza dell'assegnamento parziale $\{WA = \text{rosso}, NSW = \text{rosso}\}$ nella Figura 6.1. Notate che le variabili SA , NT e Q sono effettivamente collegate dal vincolo *Tuttediverse* dato che ogni coppia dev'essere di un colore diverso. Dopo aver applicato AC-3 all'assegnamento parziale, i domini di SA , NT e Q sono tutti ridotti a $\{\text{verde, blu}\}$. Abbiamo tre variabili e due soli colori, quindi il vincolo *Tuttediverse* è violato: una semplice procedura applicata a un vincolo di ordine superiore può rivelarsi più efficace del calcolo della consistenza d'arco a un insieme equivalente di vincoli binari.

vincolo sulle risorse

Un altro importante vincolo di ordine superiore è il **vincolo sulle risorse**, talvolta chiamato *atmost* (“al massimo”). Per esempio, supponiamo che P_1, \dots, P_4 indichino il numero di persone assegnate a quattro diverse attività. Il vincolo che non siano assegnate più di 10 persone in totale si scrive *atmost* $(10, P_1, P_2, P_3, P_4)$. Possiamo rilevare un'inconsistenza semplicemente controllando la somma dei valori minimi dei domini correnti; se per esempio ogni variabile ha il dominio $\{3, 4, 5, 6\}$, allora il vincolo *atmost* non può essere soddisfatto. Possiamo assicurare la consistenza cancellando il valore massimo di un dominio se non è consistente con i valori minimi degli altri: così, se ogni variabile del nostro esempio ha il dominio $\{2, 3, 4, 5, 6\}$, i valori 5 e 6 possono essere eliminati dai domini di tutte le variabili.

Per grandi problemi a valori interi che coinvolgono risorse limitate, come i problemi logistici che richiedono di spostare migliaia di persone con centinaia di veicoli, normalmente è impossibile rappresentare il dominio di ogni variabile come un grande insieme di interi per poi ridurlo gradualmente con i metodi di verifica della consistenza. In questi casi i domini sono rappresentati invece dagli estremi superiore e inferiore dell'intervallo di valori consentiti e sono gestiti mediante la **propagazione degli estremi** stessi. Per esempio, supponiamo che in un problema di programmazione delle linee aeree ci siano due voli, F_1 e F_2 , i cui aerei abbiano rispettivamente una capacità di 165 e 385 passeggeri. I domini iniziali per il numero dei passeggeri sui voli F_1 e F_2 saranno allora:

$$D_1 = [0, 165] \text{ e } D_2 = [0, 385].$$

propagazione degli estremi

	1	2	3	4	5	6	7	8	9
A			3		2	6			
B	9			3	5			1	
C			1	8	6	4			
D			8	1	2	9			
E	7						8		
F			6	7	8	2			
G			2	6	9	5			
H	8			2	3			9	
I			5	1	3				

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figura 6.4 (a) Un esempio di Sudoku e (b) la sua soluzione.

Supponiamo ora di avere un vincolo aggiuntivo, e cioè che i due voli insieme debbano portare esattamente 420 passeggeri: $F_1 + F_2 = 420$. Propagando i vincoli sugli estremi, possiamo ridurre i domini a:

$$D_1 = [35, 165] \text{ e } D_2 = [255, 385].$$

Diciamo che un CSP ha gli **estremi consistenti** se per ogni variabile X , e per entrambi i suoi estremi inferiore e superiore, esiste qualche valore di Y che soddisfa il vincolo tra X e Y , per ogni variabile Y . Questo tipo di propagazione degli estremi è ampiamente usato nella pratica per i CSP.

estremi consistenti

6.2.6 Sudoku

Il noto gioco **Sudoku** ha introdotto milioni di persone ai problemi di soddisfacimento di vincoli, anche se non se ne accorgono. Una griglia di Sudoku è formata da 81 caselle quadrate, alcune delle quali contengono già cifre da 1 a 9. Il gioco consiste nel riempire tutte le caselle rimanenti facendo in modo che nessuna cifra compaia due volte in una riga, colonna o riquadro 3×3 (Figura 6.4). Una riga, una colonna o un riquadro si chiama **unità**.

Sudoku

I Sudoku pubblicati su quotidiani e riviste di enigmistica hanno la proprietà di avere esattamente una soluzione. Alcuni possono essere difficili da risolvere a mano e richiedere decine di minuti, ma un risolutore CSP è in grado di risolvere migliaia di giochi al minuto.

Una istanza di Sudoku può essere considerata un CSP con 81 variabili, una per ogni casella. Utilizziamo i nomi di variabili da $A1$ ad $A9$ per la riga più in alto (da sinistra a destra), fino a $I1 - I9$ per la riga più in basso. Le caselle vuote hanno il dominio $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ e quelle già riempite hanno un dominio costituito da un singolo valore. In più, vi sono 27 diversi vincoli *Tuttediverse*: uno per ciascuna riga, colonna e riquadro di 9 caselle.

Tuttediverse(A1, A2, A3, A4, A5, A6, A7, A8, A9)

Tuttediverse(B1, B2, B3, B4, B5, B6, B7, B8, B9)

...

Tuttediverse(A1, B1, C1, D1, E1, F1, G1, H1, I1)

Tuttediverse(A2, B2, C2, D2, E2, F2, G2, H2, I2)

...

Tuttediverse(A1, A2, A3, B1, B2, B3, C1, C2, C3)

Tuttediverse(A4, A5, A6, B4, B5, B6, C4, C5, C6)

...

Vediamo fin dove possiamo arrivare con la consistenza d’arco. Ipotizziamo che i vincoli *Tuttediverse* siano stati espansi in vincoli binari (come $A1 \neq A2$) in modo che possiamo applicare direttamente l’algoritmo AC-3. Consideriamo la variabile $E6$ della Figura 6.4(a), la casella vuota tra il 2 e l’8 nel riquadro centrale. Per i vincoli nel riquadro, possiamo rimuovere 1, 2, 7 e 8 dal dominio di $E6$. Per i vincoli della colonna, possiamo eliminare 5, 6, 2, 8, 9 e 3 (anche se 2 e 8 erano già stati rimossi). Il dominio $E6$ diventa quindi {4}; in altre parole, conosciamo la soluzione per $E6$. Ora consideriamo la variabile $I6$, la casella nel riquadro in basso al centro circondata da 1, 3 e 3. Applicando la consistenza d’arco nella colonna corrispondente eliminiamo 5, 6, 2, 4 (poiché ora sappiamo che $E6$ deve essere 4), 8, 9 e 3. Eliminiamo 1 per consistenza d’arco con $I5$ e ci rimane solo il valore 7 nel dominio di $I6$. Ora ci sono 8 valori noti nella colonna 6, perciò la consistenza d’arco può inferire che $A6$ deve essere 1. L’inferenza continua lungo queste linee e alla fine AC-3 è in grado di risolvere l’intero gioco: per tutte le variabili il dominio si riduce a un singolo valore, come si vede nella Figura 6.4(b).

Naturalmente il Sudoku perderebbe il suo interesse se ogni partita potesse essere risolta da un’applicazione meccanica di AC-3, e infatti questo algoritmo funziona soltanto per gli schemi più facili. Schemi un po’ più difficili possono essere risolti da PC-2 (un algoritmo per valutare la consistenza di cammino), ma a un costo computazionale maggiore: ci sono 255.960 vincoli di cammino diversi da considerare in una partita di Sudoku. Per risolvere gli schemi più difficili e per ottenere un progresso efficiente, dovremo essere più abili.

In ogni caso, la bellezza del Sudoku per le persone umane sta nella necessità di essere abili nell’applicare strategie di inferenza più complesse. Gli appassionati danno a queste strategie nomi pittoreschi quali “naked triples” (letteralmente “triplette nude”). Questa strategia funziona così: in una qualsiasi unità (riga, colonna o riquadro), trovate tre caselle che abbiano ognuna un dominio contenente gli stessi tre numeri o un sottoinsieme di questi. Per esempio, i tre domini potrebbero essere {1, 8}, {3, 8} e {1, 3, 8}. Da qui non sappiamo quale casella contiene 1, 3 o 8, ma sappiamo che i tre numeri devono essere distribuiti fra le tre caselle. Perciò possiamo rimuovere 1, 3 e 8 dai domini di ogni *altra* casella dell’unità.

È interessante notare fin dove possiamo arrivare senza dire molto di specifico per il Sudoku. Naturalmente dobbiamo dire che ci sono 81 variabili, che i loro domini sono le cifre da 1 a 9 e che vi sono 27 vincoli *Tuttediverse*, ma al di là di questo, tutte le strategie (consistenza d’arco, consistenza di cammino e così via) si applicano in generale a tutti i CSP, non solo al problema del Sudoku. Anche “naked triples” in realtà è una strategia per forzare la consistenza di vincoli *Tuttediverse* e non è specifica del Sudoku *in sé*. Qui si rivela la potenza del formalismo CSP: per ogni nuova area di problemi, ci basta definire il problema in termini di vincoli, dopodiché possiamo affidarci ai meccanismi generali di risoluzione di vincoli.

6.3 Ricerca con backtracking per CSP

A volte, dopo la conclusione del processo di propagazione dei vincoli rimangono ancora variabili con più valori possibili. In quei casi dobbiamo effettuare una **ricerca** della soluzione. In questo paragrafo esaminiamo algoritmi di ricerca con backtracking che operano su assegnamenti parziali; nel paragrafo seguente vedremo algoritmi di ricerca locali su assegnamenti completi.

Potremmo applicare una ricerca a profondità limitata standard (cfr. il Capitolo 3). Uno stato sarebbe un assegnamento parziale e un’azione estenderebbe tale assegnamento aggiungendo, per esempio, $NSW = \text{rosso}$ o $SA = \text{blu}$ per il problema di colorare la mappa dell’Australia. Per un CSP con n variabili con dominio di dimensione d finiremmo per avere un albero di ricerca in cui tutti gli assegnamenti completi (e quindi tutte le soluzioni) sono nodi foglia di profondità n . Ma notiamo che il fattore di ramificazione al primo livello sa-

```

function RICERCA-BACKTRACKING(csp) returns una soluzione, o fallimento
  return BACKTRACKING(csp, {})

function BACKTRACKING(csp, assegnamento) returns una soluzione, o fallimento
  if assegnamento è completo then return assegnamento
  var  $\leftarrow$  SCEGLI-VARIABILE-NON-ASSEGNATA(csp, assegnamento)
  for each valore in ORDINA-VALORI-DOMINIO(csp, var, assegnamento) do
    if valore è consistente con assegnamento then
      aggiungi {var = valore} ad assegnamento
      inferenze  $\leftarrow$  INFERENZA(csp, var, assegnamento)
      if inferenze  $\neq$  fallimento then
        aggiungi inferenze a csp
        risultato  $\leftarrow$  BACKTRACKING(csp, assegnamento)
        if risultato  $\neq$  fallimento then return risultato
        rimuovi inferenze da csp
        rimuovi {var = valore} da assegnamento
    return fallimento

```

Figura 6.5 Un semplice algoritmo di backtracking per problemi di soddisfacimento di vincoli. L'algoritmo è modellato sulla ricerca in profondità ricorsiva presentata nel Capitolo 3. Le funzioni SCEGLI-VARIABILE-NON-ASSEGNATA e ORDINA-VALORI-DOMINIO implementano le euristiche di uso generale discusse nel Paragrafo 6.3.1. La funzione INFERENZA può essere usata, facoltativamente, per imporre consistenza d'arco, di cammino o *k*-consistenza, come si preferisce. Se la scelta di un valore porta al fallimento (notato da INFERENZA o BACKTRACKING), allora gli assegnamenti di valori (inclusi quelli effettuati da INFERENZA) sono annullati e si prova con un nuovo valore.

rebbe nd , perché uno qualsiasi dei d valori può essere assegnato a ognuna delle n variabili. Al livello successivo, il fattore di ramificazione è $(n - 1)d$, e così via per n livelli. Anche se i possibili assegnamenti completi sono solo d^n , l'albero generato ha $n! \cdot d^n$ foglie!

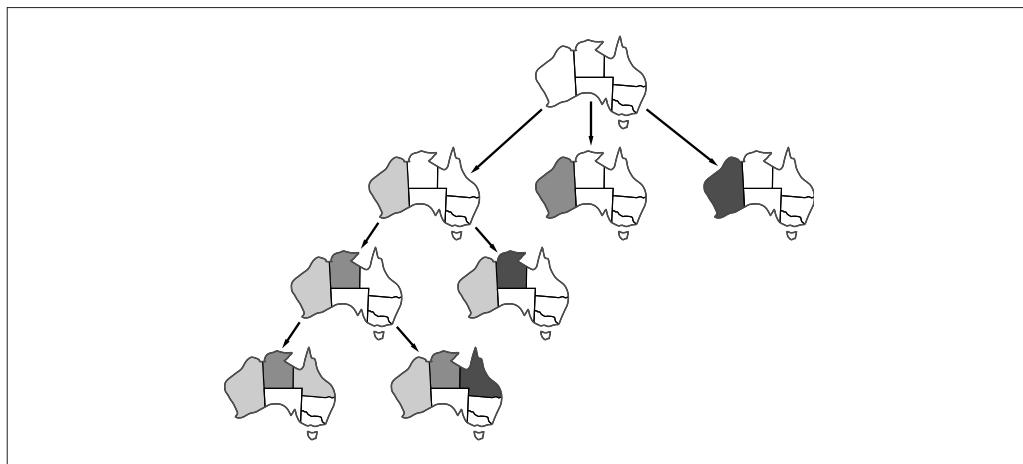
Possiamo eliminare il fattore $n!$ sfruttando una proprietà fondamentale dei CSP: la **commutatività**. Un problema è commutativo se l'ordine di applicazione di un qualsiasi insieme di azioni non conta. Nei CSP non fa differenza se assegniamo prima *NSW = rosso* e poi *SA = blu*, o vice versa. Ci basta quindi considerare *una sola* variabile in ogni nodo dell'albero di ricerca. Nella radice dell'albero potremmo scegliere tra *SA = rosso*, *SA = verde* e *SA = blu*, ma non dovremo mai scegliere tra *SA = rosso* e *WA = blu*. Con questa restrizione il numero di foglie risulta d^n , come speravamo. A ogni livello dell'albero dobbiamo scegliere quale variabile utilizzare, ma non dovremo mai effettuare il backtracking su quella scelta.

La Figura 6.5 mostra una procedura di ricerca con backtracking per problemi CSP. L'algoritmo sceglie ripetutamente una variabile non assegnata e poi prova uno a uno tutti i valori del suo dominio, cercando di estendere ogni assegnamento a una soluzione attraverso una chiamata ricorsiva. Se la chiamata ha successo, viene restituita la soluzione, mentre se fallisce, l'assegnamento è riportato allo stato precedente e si prova il valore successivo. Se non si trova alcun valore che funziona, viene restituito *fallimento*. Una parte dell'albero di ricerca per il problema della mappa australiana è mostrata nella Figura 6.6, in cui le variabili sono state assegnate nell'ordine *WA, NT, Q, ...*

Notate che RICERCA-BACKTRACKING mantiene soltanto una singola rappresentazione di uno stato (assegnamento) e altera tale rappresentazione anziché crearne di nuove (cfr. Paragrafo 3.4.3).

Figura 6.6

Una parte dell’albero di ricerca per il problema di colorazione della mappa della Figura 6.1.



Mentre gli algoritmi di ricerca non informata del Capitolo 3 potevano essere migliorati soltanto fornendo funzioni euristiche *specifiche del dominio*, la ricerca con backtracking può essere migliorata usando euristiche *indipendenti dal dominio* che traggono vantaggio dalla rappresentazione fattorizzata dei CSP. Nei sottoparagrafi seguenti mostreremo come farlo.

1. (Paragrafo 6.3.1) Quale variabile assegnare nel passo successivo (SCEGLI-VARIABILE-NON-ASSEGNATA), e in quale ordine provare i possibili valori (ORDINA-VALORI-DOMINIO)?
2. (Paragrafo 6.3.2) Quali inferenze vanno eseguite a ciascun passaggio della ricerca (INFERENZA)?
3. (Paragrafo 6.3.3) Possiamo fare backtracking di più passi, quando appropriato (BACKTRACK)?
4. (Paragrafo 6.3.4) Possiamo salvare e riutilizzare risultati parziali della ricerca?

6.3.1 Ordinamento di variabili e valori

L’algoritmo di ricerca con backtracking contiene la riga:

var \leftarrow SCEGLI-VARIABILE-NON-ASSEGNATA(*csp, assegnamento*) .

La strategia più semplice per SCEGLI-VARIABILE-NON-ASSEGNATA è l’ordinamento statico: scegliere le variabili nell’ordine $\{X_1, X_2, \dots\}$. La seconda strategia più semplice è quella di scegliere le variabili a caso. Nessuna delle due strategie è ottima. Per esempio, dopo gli assegnamenti $WA = \text{rosso}$ e $NT = \text{verde}$ nella Figura 6.6 c’è un solo valore possibile per SA , per cui la cosa più sensata come passo successivo sarebbe assegnare $SA = \text{blu}$ invece di occuparsi dell’assegnamento di Q . In effetti, dopo l’assegnamento di SA le scelte per Q , NSW e V sono tutte forzate.

euristica MRV

L’idea di scegliere la variabile con il minor numero di valori “legali” è chiamata **euristica MRV** (*minimum remaining values*): è stata chiamata anche euristica “della variabile più vincolata” o *fail-first*, perché sceglie la variabile per cui è maggiore la probabilità di arrivare presto a un fallimento, potando così l’albero di ricerca. Se una variabile X non ha più alcun possibile valore, l’euristica MRV sceglierà X e il fallimento sarà rilevato immediatamente, evitando così ricerche inutili su altre variabili. L’euristica MRV solitamente ha prestazioni migliori di un ordinamento casuale o statico, talvolta di vari ordini di grandezza, anche se i risultati variano notevolmente a seconda del problema.

L’euristica MRV non è di alcun aiuto nella scelta della prima regione da colorare nel caso dell’Australia, perché all’inizio ognuna di esse ha esattamente tre colori legali. In questo ca-

so, viene in aiuto l'**euristica di grado** (*degree heuristic*). Quest'euristica cerca di ridurre il fattore di ramificazione delle scelte future scegliendo la variabile coinvolta nel maggior numero di vincoli con le altre variabili non assegnate. Nella Figura 6.1, *SA* è la variabile di grado più alto, 5; le altre variabili hanno grado 2 o 3, eccetto *T*, che è di grado 0. Se assegniamo prima *SA*, applicando l'euristica di grado potremmo risolvere il problema senza passi falsi, cioè arrivare a una soluzione senza backtracking.. L'euristica MRV solitamente è più potente, ma quella di grado può tornare utile nel caso si verifichino dei "pareggi" nell'ordinamento.

Una volta scelta una variabile, l'algoritmo deve decidere l'ordine con cui esaminare i suoi possibili valori. Per far questo, risulta efficace l'euristica del **valore meno vincolante**, che predilige il valore che lascia più libertà alle variabili adiacenti sul grafo dei vincoli. Per esempio, supponete che nella Figura 6.1 abbiamo generato l'assegnamento parziale *WA = rosso* e *NT = verde*, e che la nostra prossima scelta riguardi *Q*. Il blu sarebbe una cattiva scelta, perché elimina l'ultimo valore legale per il vicino di *Q*, *SA*. L'euristica del valore meno vincolante preferisce quindi il colore rosso. In generale, l'euristica cerca sempre di lasciare la massima flessibilità ai successivi assegnamenti di variabili.

Perché la selezione di una variabile dovrebbe essere di tipo *fail-first* ("prima quelle che falliscono") mentre la selezione di un valore dovrebbe essere di tipo *fail-last* ("per ultimi quelli che falliscono")? Perché ogni variabile prima o poi dovrà essere assegnata, perciò scegliendo quelle che hanno più probabilità di fallire per prime, in media avremo meno assegnamenti di successo su cui eseguire il backtracking. Per l'ordinamento di valori, il trucco è che ci serve una sola soluzione, perciò ha senso cercare prima i valori più probabili. Se volessimo enumerare tutte le soluzioni, anziché limitarci a trovarne una, l'ordinamento di valori non servirebbe.

euristica di grado

valore meno vincolante

6.3.2 Alternanza di ricerca e inferenza

Abbiamo visto come AC-3 e altri algoritmi possano ridurre i domini di variabili *prima* di iniziare la ricerca. Ma l'inferenza può risultare ancora più potente *nel corso* di una ricerca: ogni volta che scegliamo un valore per una variabile, abbiamo una opportunità del tutto nuova di inferire nuove riduzioni dei domini delle variabili adiacenti.

Una delle più semplici forme di inferenza è la cosiddetta **verifica in avanti** (*forward checking*). Ogni volta che una variabile *X* è assegnata, il processo di verifica in avanti stabilisce la consistenza d'arco per essa: per ogni variabile non assegnata *Y* collegata a *X* da un vincolo, cancella dal dominio di *Y* ogni valore non consistente con quello scelto per *X*.

verifica in avanti

La Figura 6.7 mostra il progresso di una ricerca con backtracking sul CSP dell'Australia con verifica in avanti. Ci sono due punti da notare in quest'esempio. Prima di tutto, notate

	WA	NT	Q	NSW	V	SA	T
domini iniziali	R V B	R V B	R V B	R V B	R V B	R V B	R V B
dopo WA = rosso	R	V B	R V B	R V B	R V B	V B	R V B
dopo Q = verde	R	B	V	R B	R V B	B	R V B
dopo V = blu	R	B	V	R	B		R V B

Figura 6.7 Il progresso di una ricerca per la colorazione una mappa che sfrutta la verifica in avanti. Prima di tutto si assegna *WA = rosso*; fatto questo, la verifica in avanti cancella il colore rosso dai domini delle variabili adiacenti *NT* e *SA*. Dopo *Q = verde*, tale colore è rimosso dai domini di *NT*, *SA* e *NSW*. Dopo *V = blu*, anche *blu* è cancellato dai domini di *NSW* e *SA*, dimodoché *SA* non ha più valori legali.

che dopo aver assegnato $WA = \text{rosso}$ e $Q = \text{verde}$, i domini di NT e SA sono ridotti a un solo valore; abbiamo eliminato completamente la ramificazione su queste variabili propagando informazione da WA e Q . Un secondo aspetto da notare è che, dopo l’assegnamento $V = \text{blu}$, il dominio di SA è vuoto. Ne consegue che la verifica in avanti ha rilevato che l’assegnamento parziale $\{WA = \text{rosso}, Q = \text{verde}, V = \text{blu}\}$ è inconsistente con i vincoli del problema, e l’algoritmo potrà quindi tornare immediatamente indietro con il backtracking.

Per molti problemi la ricerca risulterà più efficace se combiniamo l’euristica MRV con la verifica in avanti. Considerate la Figura 6.7 dopo l’assegnamento $\{WA = \text{rosso}\}$. Intuitivamente sembra che l’assegnamento vincoli le variabili adiacenti, NT e SA , perciò dobbiamo considerare prima queste variabili, e poi tutte le altre andranno a posto. È esattamente ciò che accade con MRV: NT e SA hanno due valori, perciò viene scelta prima una di esse, poi l’altra, poi Q , NSW e V , nell’ordine. Alla fine T ha ancora tre valori, e uno qualsiasi di essi funziona. Possiamo considerare la verifica in avanti come un modo efficiente per calcolare in modo incrementale le informazioni di cui l’euristica MRV necessita per svolgere il proprio compito.

MAC La verifica in avanti rileva molte inconsistenze, ma non tutte. Il problema è non guarda abbastanza avanti. Per esempio, considerate la riga della Figura 6.7 con $Q = \text{verde}$. Abbiamo reso WA e Q arco-consistenti, ma rimangono sia NT che SA per cui l’unico valore possibile è blu , quindi c’è un’inconsistenza, dato che sono vicini. L’algoritmo **MAC**, dall’inglese *maintaining arc consistency*, rileva inconsistenze come questa. Dopo che a una variabile X_i è assegnato un valore, la procedura INFERENZA richiama AC-3, ma invece di una coda di tutti gli archi del CSP, iniziamo soltanto con gli archi (X_i, X_j) per tutte le X_j che sono variabili non assegnate adiacenti a X_i . Da qui, AC-3 esegue la propagazione dei vincoli come di consueto, e se una variabile vede il proprio dominio ridursi all’insieme vuoto, la chiamata di AC-3 ritorna un fallimento e sappiamo di dover effettuare immediatamente il backtracking. Possiamo vedere che MAC è strettamente più potente della verifica in avanti perché quest’ultima procede come MAC sugli archi iniziali nella coda di MAC, ma a differenza di questo, la verifica in avanti non propaga ricorsivamente i vincoli quando si apportano modifiche ai domini delle variabili.

6.3.3 Backtracking intelligente: guardarsi indietro

L’algoritmo RICERCA-BACKTRACKING della Figura 6.5 ha una politica molto semplice riguardo a cosa fare quando un ramo della ricerca fallisce: tornare indietro alla variabile precedente e provare a usare un valore diverso. Questo si chiama **backtracking cronologico**, perché viene rivisitato il punto decisionale *più recente*. In questo sottoparagrafo considereremo alcune possibilità migliori.

Considerate ciò che succede nella Figura 6.1 quando applichiamo un backtracking semplice con ordinamento fisso delle variabili Q, NSW, V, T, SA, WA, NT . Supponiamo di aver generato l’assegnamento parziale $\{Q = \text{rosso}, NSW = \text{verde}, V = \text{blu}, T = \text{rosso}\}$. Quando prendiamo la variabile successiva, SA , appuriamo che ogni possibile valore viola un vincolo. Allora torniamo indietro in T e assegniamo un colore diverso alla Tasmania! Palesemente questa è un’azione alquanto scioccante: cambiare il colore della Tasmania non può tornare utile per risolvere un problema in South Australia.

Un approccio più intelligente è quello di effettuare il backtracking risalendo fino a una delle variabili che ha causato il fallimento. Per fare ciò, memorizziamo un insieme di assegnamenti che sono in conflitto con qualche valore di SA . Questo insieme è chiamato **insieme dei conflitti**; in questo caso l’insieme dei conflitti per SA è $\{Q = \text{rosso}, NSW = \text{verde}, V = \text{blu}\}$. Il metodo del **backjumping** (letteralmente, “salto all’indietro”) torna indietro fino all’assegnamento *più recente* nell’insieme dei conflitti; in questo caso saltando la Tasmania e cercando un altro valore per V . Si può facilmente implementare tutto ciò modificando

backtracking cronologico

insieme dei conflitti

backjumping

RICERCA-BACKTRACKING in modo che memorizzi l'insieme dei conflitti mentre cerca un valore legale da assegnare. Se non trova alcun valore legale, insieme all'indicazione del fallimento dovrà restituire l'elemento più recente dell'insieme dei conflitti.

I lettori più attenti avranno già notato che la verifica in avanti può fornire l'insieme dei conflitti senza lavoro aggiuntivo: ogni volta che la verifica in avanti causata da un assegnamento $X = x$ fa sì che venga cancellato un valore dal dominio di Y , $X = x$ dev'essere aggiunto all'insieme dei conflitti di Y . Se viene cancellato l'ultimo valore dal dominio di Y , tutti gli assegnamenti dell'insieme dei conflitti di Y vanno aggiunti all'insieme dei conflitti di X . In questo modo, ora sappiamo che $X = x$ porta a una contraddizione (in Y) e quindi che occorre provare un diverso assegnamento per X .

I lettori *veramente* attenti avranno notato qualcosa di strano: il backjumping si verifica quando ogni valore in un dominio configge con l'assegnamento corrente; ma la verifica in avanti rileva questo evento e impedisce alla ricerca di raggiungere quel nodo in ogni caso! In effetti, si può dimostrare che *ogni* ramo dell'albero potato dal backjumping è parimenti potato dalla verifica in avanti. Di conseguenza, il semplice backjumping è ridondante in una ricerca con verifica in avanti, o quando si usano verifiche di consistenza più potenti come MAC – basta l'uno o l'altro metodo.

Nonostante questa osservazione, il backjumping rimane una buona idea: vale la pena di basare il processo di backtracking sulle ragioni del fallimento. Il backjumping rileva un fallimento quando il dominio di una variabile diventa vuoto, ma in molti casi un ramo è “condannato” molto prima che questo si verifichi. Consideriamo ancora l'assegnamento parziale inconsistente $\{WA = \text{rosso}, NSW = \text{rosso}\}$. Supponiamo di provare come passo successivo $T = \text{rosso}$ e poi assegnare valori alle variabili NT, Q, V, SA . Sappiamo che per queste ultime quattro non c'è un possibile assegnamento, per cui a un certo punto esauriremo i valori per NT . A questo punto, la domanda è: dove ritornare con il backtracking? Il backjumping non può funzionare, perché NT ha *effettivamente* valori consistenti con le variabili assegnate in precedenza: NT non ha un insieme dei conflitti completo delle precedenti variabili che hanno causato il suo fallimento. Sappiamo comunque che le quattro variabili NT, Q, V e SA , *prese insieme*, falliscono a causa di un insieme delle variabili precedenti, che devono essere quelle che configgono direttamente con queste quattro.

Questo ci porta a una diversa e più profonda nozione di insieme dei conflitti per una variabile come NT : è l'insieme delle variabili precedenti che ha fatto sì che NT , *insieme alle variabili successive*, non avesse alcuna soluzione consistente. In questo caso l'insieme è composto da WA e NSW , quindi l'algoritmo dovrebbe ritornare a NSW saltando la Tasmania. Un algoritmo di backjumping che sfrutta insiemi dei conflitti definiti in questo modo è chiamato **backjumping guidato dai conflitti**.

Ora dobbiamo spiegare come calcolare questi nuovi insiemi dei conflitti. In effetti, il metodo è piuttosto semplice. Il fallimento “terminale” di un ramo della ricerca si verifica sempre perché il dominio di una variabile diventa vuoto; quella variabile ha un insieme dei conflitti standard. Nel nostro esempio, SA fallisce, e il suo insieme dei conflitti è (poniamo) $\{WA, NT, Q\}$. Saltiamo indietro fino a Q , che *assorbe* l'insieme dei conflitti di SA (tranne se stesso, naturalmente) integrandolo nel suo insieme dei conflitti diretti, che è $\{NT, NSW\}$; il nuovo insieme è $\{WA, NT, NSW\}$. Questo significa che non c'è soluzione da Q in avanti, dati i precedenti assegnamenti a $\{WA, NT, NSW\}$. Di conseguenza, torniamo indietro con il backtracking fino a NT , il più recente tra questi. NT assorbe $\{WA, NT, NSW\} - \{NT\}$ nel suo insieme dei conflitti diretti $\{WA\}$, e il risultato è $\{WA, NSW\}$. Adesso l'algoritmo salta indietro fino a NSW , come ci saremmo augurati. Per riassumere: sia X_j la variabile corrente e $\text{conf}(X_j)$ il suo insieme dei conflitti. Se ogni possibile valore di X_j fallisce, si salta indietro (*backjump*) alla variabile più recente X_i in $\text{conf}(X_j)$, e si assegna.

$$\text{conf}(X_i) \leftarrow \text{conf}(X_i) \cup \text{conf}(X_j) - \{X_i\} .$$

**backjumping
guidato dai conflitti**

apprendimento dei
vincoli
no-good

6.3.4 Apprendimento dei vincoli

Quando raggiungiamo una contraddizione, il backjumping ci può indicare fino a dove risalire, in modo da non perdere tempo a modificare variabili che non risolveranno il problema. Ma vorremmo anche evitare di imbatterci nuovamente nello stesso problema. Quando la ricerca arriva a una contraddizione, sappiamo che un sottoinsieme dell'insieme dei conflitti è il responsabile. L'**apprendimento dei vincoli** è l'idea di trovare un insieme minimo di variabili dell'insieme dei conflitti che causa il problema. Questo insieme di variabili, insieme ai valori corrispondenti, si chiama **no-good** (letteralmente “buono a nulla”). Ora registriamo il no-good, aggiungendo un nuovo vincolo al CSP per proibire questa combinazione di assegnamenti, o mantenendo una cache separata di no-good.

Per esempio, consideriamo lo stato $\{WA = \text{rosso}, NT = \text{verde}, Q = \text{blu}\}$ nella riga inferiore della Figura 6.6. La verifica in avanti può indicarci che questo stato è un no-good perché non esiste un assegnamento valido per SA . In questo caso particolare, registrare il no-good non servirebbe, perché una volta che potiamo questo ramo dall'albero di ricerca, non incontreremo più questa combinazione. Ma supponiamo che l'albero di ricerca della Figura 6.6 fosse parte in realtà di un albero di ricerca più grande, iniziato con l'assegnamento di valori per V e T . Allora sarebbe utile registrare $\{WA = \text{rosso}, NT = \text{verde}, Q = \text{blu}\}$ come no-good perché incontreremmo di nuovo lo stesso problema per ogni possibile insieme di assegnamenti di V e T .

I no-good possono essere utilizzati efficacemente dalla verifica in avanti o dal backjumping. L'apprendimento dei vincoli è una delle più importanti tecniche utilizzate dai moderni risolutori di CSP per ottenere l'efficienza su problemi complessi.

6.4 Ricerca locale per CSP

Gli algoritmi di ricerca locale (cfr. Paragrafo 4.1) si rivelano molto efficaci per la soluzione di svariate tipologie di CSP. Utilizzano una formulazione a stato completo (descritta nel Paragrafo 4.1.1) in cui ogni stato assegna un valore a ogni variabile e la ricerca cambia il valore di una variabile per volta. Come esempio possiamo fare riferimento al problema delle 8 regine, definito come CSP nel Paragrafo 6.1.3. Nella Figura 6.8 iniziamo a sinistra con un assegnamento completo delle 8 variabili; generalmente questo viola diversi vincoli. Poi scegliamo a caso una variabile in conflitto, che in questo caso è Q_8 , la colonna più a destra. Vorremmo cambiarne il valore in modo da arrivare più vicino a una soluzione; l'approccio

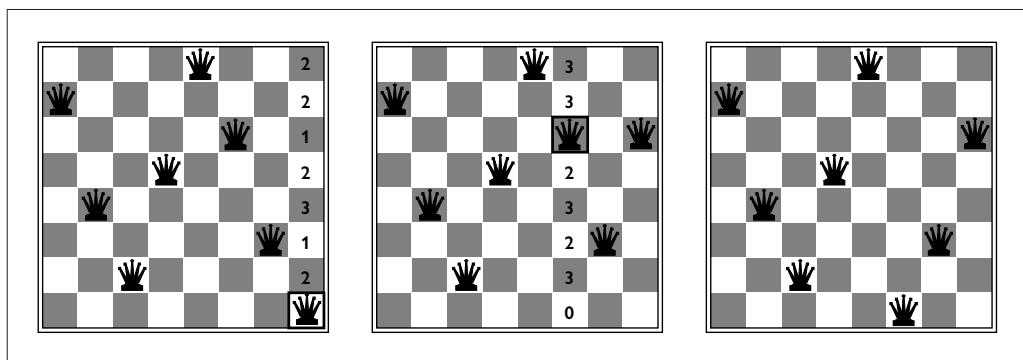


Figura 6.8 Una soluzione in due passi per un problema delle 8 regine, ottenuta con min-conflicts. A ogni passo una regina viene scelta e riposizionata nella sua colonna. Il numero di conflitti (in questo caso, il numero delle regine attaccanti) è indicato in ogni casella. L'algoritmo sposta la regina nella casella corrispondente al numero minimo di conflitti, risolvendo casualmente le situazioni di parità.

```

function MIN-CONFLICTS(csp, max_passi) returns una soluzione, o fallimento
  inputs: csp, un problema di soddisfacimento di vincoli
           max_passi, il numero di passi consentiti prima di abbandonare

  corrente  $\leftarrow$  un assegnamento completo iniziale per csp
  for i = 1 to max_passi do
    if corrente è una soluzione di csp then return corrente
    var  $\leftarrow$  una variabile in conflitto, scelta a caso in csp.VARIABILI
    valore  $\leftarrow$  il valore v di var che minimizza CONFLITTI(csp, var, v, corrente)
    aggiorna var = valore in corrente
  return fallimento

```

Figura 6.9 L'algoritmo di ricerca locale MIN-CONFLICTS per la risoluzione dei CSP. Lo stato iniziale può essere scelto a caso o con un processo di assegnamento greedy che sceglie il valore che genera il minimo numero di conflitti per ogni variabile. La funzione CONFLITTI, dato l'assegnamento corrente, conta il numero di vincoli violati da un particolare valore, dato l'assegnamento corrente.

più ovvio è scegliere il valore che porta al minimo numero di conflitti con le altre variabili: questa è l'euristica **min-conflicts**.

Nella Figura 6.8 vediamo che ci sono due righe che violano un solo vincolo; scegliamo $Q_8 = 3$ (cioè spostiamo la regina nella colonna 8, riga 3). All'iterazione successiva, nella scacchiera al centro della figura, selezioniamo Q_6 come variabile da cambiare e notiamo che spostare la regina nella riga 8 non genera alcun conflitto. A questo punto non ci sono più variabili in conflitto, perciò abbiamo una soluzione. L'algoritmo è riportato nella Figura 6.9.²

L'euristica min-conflicts è sorprendentemente efficace per molti CSP. Ciò che è davvero stupefacente, nel problema a *n* regine, è che se non si conta il piazzamento iniziale dei pezzi il tempo di esecuzione di min-conflicts è quasi *indipendente dalle dimensioni del problema*. Min-conflicts è capace di risolvere anche il problema con *un milione* di regine in una media di 50 passi (dopo l'assegnamento iniziale). Quest'osservazione notevole è stata lo sprone che negli anni 1990 ha portato a una grande mole di studi sulla ricerca locale e sulla distinzione tra problemi facili e difficili, che tratteremo nel Paragrafo 7.6.3. A grandi linee, si può dire che il problema delle *n* regine è facile per la ricerca locale perché le soluzioni sono distribuite densamente nello spazio degli stati. Min-conflicts peraltro funziona bene anche su problemi difficili: è stato usato per pianificare le osservazioni del telescopio Hubble, riducendo il tempo richiesto per programmare una settimana di osservazioni da tre settimane (!) a circa 10 minuti.

Tutte le tecniche di ricerca locale del Paragrafo 4.1 sono buone candidate per l'applicazione a problemi CSP, e alcune si sono dimostrate particolarmente efficaci. Il panorama di un CSP sotto l'euristica min-conflicts presenta solitamente una serie di plateau. Potrebbero esserci milioni di assegnamenti di variabili che distano soltanto un conflitto da una soluzione. La ricerca nel plateau, che consente mosse laterali in altri stati con lo stesso valore, può aiutare la ricerca locale a uscire da questo plateau. Questo aggirarsi sul plateau può essere indirizzato con la **ricerca tabù**: mantenere un piccolo elenco di stati visitati di recente e impe-

min-conflicts

² La ricerca locale può essere facilmente estesa a problemi di ottimizzazione di vincoli (COP, *constraint optimization problem*). In tal caso, per ottimizzare la funzione obiettivo si possono usare tutte le tecniche di hill climbing e simulated annealing.

constraint weighting

dire all'algoritmo di tornare in tali stati. Anche il simulated annealing può essere utilizzato per sfuggire dai plateau.

Un'altra tecnica, denominata **constraint weighting** (“pesatura dei vincoli”) può aiutare a concentrare la ricerca sui vincoli importanti. A ogni vincolo è assegnato un peso numerico, che inizialmente vale 1 per tutti. A ogni passo della ricerca, l'algoritmo sceglie di modificare una coppia variabile/valore che porterà al minimo peso totale di tutti i vincoli violati. I pesi sono poi aggiustati incrementando il peso di ciascun vincolo violato dall'assegnamento corrente. Si hanno così due vantaggi: si aggiunge una topografia ai plateau, assicurandosi che sia possibile migliorare dallo stato corrente, e inoltre si aggiunge l'apprendimento: nel tempo, si assegnano pesi superiori ai vincoli che si dimostrano difficili da risolvere.

Un altro vantaggio della ricerca locale è che può essere usata in un ambiente online (cfr. Paragrafo 4.5) quando il problema stesso può cambiare. Consideriamo un problema di programmazione dei voli settimanali di una linea aerea. Il programma potrebbe dover gestire in una settimana migliaia di voli e decine di migliaia di assegnamenti di personale, e il verificarsi di cattivo tempo in un solo aeroporto potrebbe rendere inattuabile la soluzione (schedule) originaria. È fondamentale essere in grado di costruire un programma alternativo con il numero minimo di cambiamenti. Questo può essere fatto facilmente da un algoritmo di ricerca locale, fornendo come stato iniziale lo schedule da modificare. Una ricerca con backtracking che prendesse in considerazione il nuovo insieme di vincoli richiederebbe molto più tempo e potrebbe trovare una soluzione molto diversa dallo schedule originario.

6.5 La struttura dei problemi

In questo paragrafo esamineremo i modi in cui si può sfruttare la *struttura* del problema, rappresentata dal grafo dei vincoli, per trovare rapidamente le soluzioni. La maggior parte degli approcci presentati si applica anche ad altri problemi oltre i CSP, per esempio al ragionamento probabilistico.

L'unico modo in cui possiamo sperare di gestire la vasta complessità del mondo reale è quello di scomporlo in sottoproblemi. Guardando ancora una volta il grafo dei vincoli per l'Australia (Figura 6.1(b), ripetuto come Figura 6.12(a)), un fatto salta agli occhi: la Tasmania non è collegata al continente.³ Intuitivamente, appare chiaro che la colorazione della Tasmania e quella del continente principale sono **sottoproblemi indipendenti** – qualsiasi soluzione per il continente, unita a una qualsiasi soluzione per la Tasmania, darà luogo a una soluzione globale per l'intera mappa.

L'indipendenza si può appurare semplicemente cercando **componenti connessi** sul grafo dei vincoli. Ogni componente corrisponde a un sottoproblema CSP_i . Se l'assegnamento S_i è una soluzione di CSP_i , allora $\bigcup_i S_i$ è una soluzione di $\bigcup_i CSP_i$. Perché questo fatto è importante? Supponiamo che ogni CSP_i abbia c variabili prese da un totale di n variabili, dove c è una costante. Allora ci sono n/c sottoproblemi, ognuno dei quali richiede al massimo d^c “lavoro” per essere risolto, dove d è la dimensione del dominio. Quindi il lavoro totale da svolgere è $O(d^c n/c)$, che cresce *linearmente* con n ; senza la scomposizione il lavoro sarebbe $O(d^n)$, che cresce esponenzialmente. Facciamo un esempio più concreto: suddividere un CSP booleano con 100 variabili in quattro sottoproblemi riduce il tempo necessario per trovare una soluzione nel caso pessimo dalla durata della vita dell'universo a meno di un secondo.

I sottoproblemi completamente indipendenti ci piacciono moltissimo; purtroppo però sono anche rari. Fortunatamente, esistono altre strutture di grafi facili da risolvere. Per esem-

sottoproblemi indipendenti**componenti connessi**

³ Un cartografo molto professionale, o un nativo della Tasmania particolarmente patriottico, potrebbero obiettare che la Tasmania non dovrebbe avere lo stesso colore della regione più vicina sulla terraferma, proprio per evitare l'impressione che *potrebbe* far parte di quello stato.

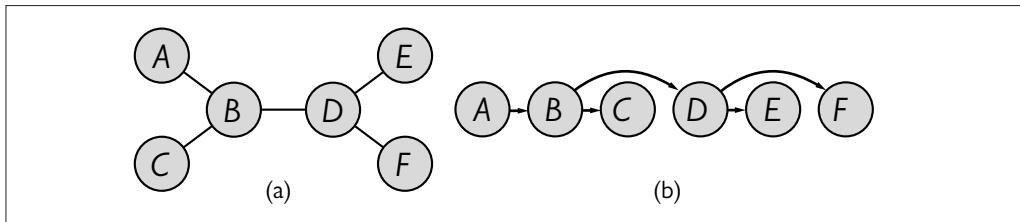


Figura 6.10 (a) Il grafo dei vincoli di un CSP con una struttura ad albero. (b) Un ordinamento lineare delle variabili consistente con l'albero, con A come radice. Si parla di **ordinamento topologico** delle variabili.

pio, un grafo dei vincoli è un **albero** quando due variabili qualsiasi sono collegate da un solo cammino. Mostreremo che ogni *CSP strutturato come un albero può essere risolto in un tempo che cresce linearmente con il numero delle variabili*.⁴ La chiave è un nuovo concetto di consistenza, denominato **consistenza d'arco orientato** o DAC (*directional arc consistency*). Un CSP si dice arco-direzionale-orientato con un ordinamento di variabili X_1, X_2, \dots, X_n se e solo se ogni X_i è arco-consistente con ogni X_j per $j > i$.

Per risolvere un CSP con struttura ad albero, per prima cosa si sceglie una variabile come radice dell'albero e si sceglie un ordinamento delle variabili tale che ogni variabile appaia dopo il proprio genitore nell'albero. Un ordinamento di questo tipo si chiama **ordinamento topologico**. La Figura 6.10(a) mostra un albero di esempio e la Figura 6.10(b) mostra un possibile ordinamento. Ogni albero con n nodi ha $n - 1$ archi, perciò possiamo rendere questo grafo arco-direzionale-orientato in $O(n)$ passi, ognuno dei quali deve confrontare fino a d possibili valori di dominio per due variabili, con un tempo totale di $O(nd^2)$. Una volta ottenuto un grafo arco-direzionale-orientato, possiamo semplicemente percorrere la lista di variabili e scegliere qualsiasi valore rimanente. Poiché ogni collegamento da un genitore al proprio figlio è arco-consistente, sappiamo che per ogni valore scelto come genitore, ci sarà un valore valido da scegliere come figlio. Ciò significa che non dovremo ricorrere al backtracking, ma possiamo spostarci linearmente tra le variabili. L'algoritmo completo è mostrato nella Figura 6.11.

Ora che abbiamo un algoritmo efficiente per gli alberi, possiamo considerare se è possibile ridurre in qualche modo grafi di vincoli generici ad alberi. Esistono due modi per fare questo: rimuovendo nodi (Paragrafo 6.5.1) oppure fondendoli tra loro (Paragrafo 6.5.2).

6.5.1 Condizionamento con insieme di taglio

Il primo modo per ridurre un grafo di vincoli a un albero prevede che si assegnino dei valori ad alcune variabili in modo che quelle rimanenti formino un albero. Considerate ancora il grafo dei vincoli per la mappa dell'Australia, mostrato ancora nella Figura 6.12(a). Senza l'Australia del Sud (SA) il grafo diventerebbe un albero, come si vede in (b). Fortunatamente possiamo eliminare l'Australia del Sud (sul grafo, non nel mondo reale) fissando un valore per SA e cancellando dai domini delle altre variabili tutti i valori che sono inconsistenti con quello.

Fatto questo, ogni soluzione per il CSP dopo la rimozione di SA e dei suoi vincoli sarà consistente col valore prescelto per SA (questo funziona nei CSP binari; la situazione è più complessa se entrano in gioco dei vincoli di ordine superiore). Possiamo quindi risolvere l'albero rimanente con l'algoritmo appena fornito e risolvere così l'intero problema. Natu-



consistenza d'arco
orientato

ordinamento
topologico

⁴ Purtroppo ben poche regioni del mondo hanno una mappa strutturata ad albero, anche se il Sulawesi ci si avvicina.

Figura 6.11

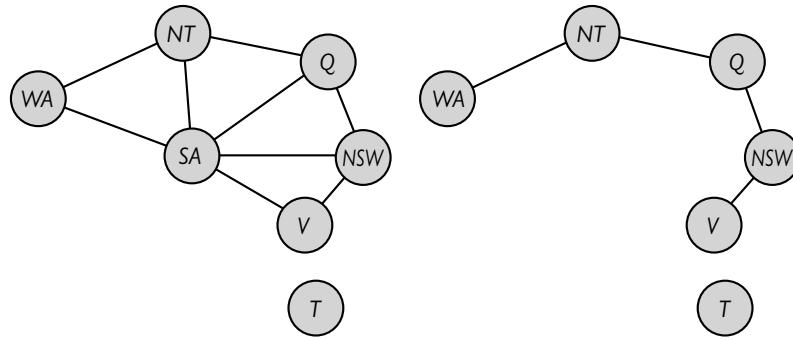
L'algoritmo RISOLUTORE-CSP-ALBERO per risolvere CSP con struttura ad albero. Se il CSP ha una soluzione, la troveremo in tempo lineare, altrimenti rileveremo una contraddizione.

```
function RISOLUTORE-CSP-ALBERO(csp) returns una soluzione, o fallimento
  inputs: csp, un CSP con componenti  $\mathcal{X}$ ,  $\mathcal{D}$ ,  $\mathcal{C}$ 

   $n \leftarrow$  numero di variabili in  $\mathcal{X}$ 
  assegnamento  $\leftarrow$  un assegnamento vuoto
  radice  $\leftarrow$  una qualsiasi variabile in  $\mathcal{X}$ 
   $\mathcal{X} \leftarrow$  OrdinamentoTopologico( $\mathcal{X}$ , radice)
  for  $j = n$  down to 2 do
    RENDI-ARCO-CONSISTENTE(PADRE( $X_j$ ),  $X_j$ )
    if non può essere reso consistente then return fallimento
  for  $i = 1$  to  $n$  do
    assegnamento[ $X_i$ ]  $\leftarrow$  qualsiasi valore consistente da  $D_i$ 
    if non esiste valore consistente then return fallimento
  return assegnamento
```

Figura 6.12

(a) Il grafo dei vincoli originale della Figura 6.1.
 (b) Il grafo dei vincoli dopo la rimozione di SA.

**insieme di taglio dei cicli**

ralmente, nel caso generale il valore scelto per *SA* potrebbe essere sbagliato (cosa che non può succedere con la colorazione di una mappa), per cui potremmo dover provare ogni valore possibile. L'algoritmo generale risultante è il seguente.

1. Scegliete un sottoinsieme *S* delle variabili del CSP tale che il grafo dei vincoli diventi un albero dopo la rimozione di *S*. *S* prende il nome di **insieme di taglio dei cicli** (*cycle cutset*).
2. Per ogni possibile assegnamento delle variabili in *S* che soddisfa tutti i vincoli su *S*,
 - rimuovete dal dominio delle variabili rimanenti tutti i valori non consistenti con gli assegnamenti in *S*, e
 - se il CSP risultante ha una soluzione, restituitela insieme all'assegnamento per *S*.

Se l'insieme di taglio dei cicli ha dimensione *c*, il tempo di esecuzione totale è $O(d^c \cdot (n - c)d^2)$: dobbiamo provare ciascuna delle d^c combinazioni di valori delle variabili in *S*, e per ogni combinazione dobbiamo risolvere un problema su un albero di dimensione $n - c$. Se il grafo è “quasi un albero” *c* sarà piccolo, e il risparmio di tempo rispetto al backtracking risulterà enorme: per il nostro esempio con 100 variabili booleane, se riuscissimo a trovare un insieme

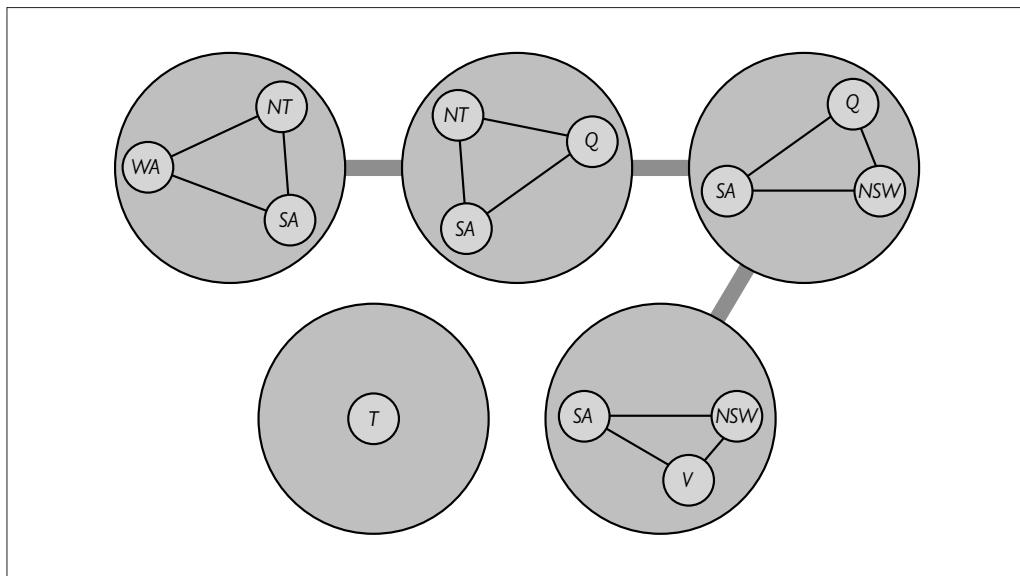


Figura 6.13
Una scomposizione ad albero del grafo dei vincoli della Figura 6.12(a).

di taglio di dimensione $c = 20$, potremmo ridurre il tempo necessario dall’intera vita dell’universo a pochi minuti. Nel caso pessimo, però, c può arrivare anche a $(n - 2)$. Trovare il *più piccolo* insieme di taglio dei cicli è un problema NP-difficile, ma si conoscono diversi algoritmi per farlo in modo approssimato ed efficiente. Questo approccio algoritmico è generalmente chiamato **condizionamento con insieme di taglio** (*cutset conditioning*) e lo incontreremo di nuovo nel Capitolo 13, in cui vedremo come usarlo nel ragionamento probabilistico.

**condizionamento
con insieme
di taglio**

6.5.2 Scomposizione ad albero

Il secondo modo per ridurre un grafo di vincoli a un albero è basato sulla costruzione di una **scomposizione ad albero** (*tree decomposition*) del grafo di vincoli: una trasformazione del grafo originario in un albero in cui ogni nodo è costituito da un insieme di variabili, come nella Figura 6.13. Una scomposizione ad albero deve soddisfare i seguenti tre requisiti:

- ogni variabile del problema originale deve comparire in almeno uno dei nodi dell’albero;
- se due variabili sono collegate da un vincolo nel problema originale, devono comparire insieme (con il vincolo) in almeno uno dei nodi dell’albero;
- se una variabile compare in due nodi nell’albero, deve essere presente anche in tutti i nodi che compongono il cammino che li collega.

**scomposizione
ad albero**

Le prime due condizioni assicurano che tutte le variabili e i vincoli siano presenti nella scomposizione. La terza sembra riguardare un aspetto tecnico, ma ci consente di affermare che ogni variabile del problema originario deve avere lo stesso valore ovunque appaia: i vincoli nell’albero dicono che una variabile in un nodo dell’albero deve avere lo stesso valore della variabile corrispondente nel nodo adiacente. Per esempio, *SA* appare in tutti i quattro nodi collegati nella Figura 6.13, perciò ogni arco nella scomposizione ad albero include il vincolo che il valore di *SA* in un nodo sia uguale al valore di *SA* nel nodo successivo. Potete verificare osservando la Figura 6.12 che questa scomposizione è valida.

Una volta che abbiamo un grafo strutturato ad albero, possiamo applicare RISOLUTORE-CSP-ALBERO per ottenere una soluzione in tempo $O(nd^2)$, dove n è il numero di nodi dell’albero e d è la dimensione del dominio più grande. Notate però che, nell’albero, un dominio è un insieme di *tuple* di valori e non di valori singoli.

Per esempio, il nodo in alto a sinistra nella Figura 6.13 rappresenta, a livello del problema originario, un sottoproblema con variabili $\{WA, NT, SA\}$, dominio $\{\text{rosso, verde, blu}\}$ e vincoli $WA \neq NT, SA \neq NT, WA \neq SA$. A livello dell’albero il nodo rappresenta una singola variabile, che possiamo chiamare $SANTWA$, il cui valore deve essere una tupla di tre colori, come $(\text{rosso, verde, blu})$, ma non $(\text{rosso, rosso, blu})$, perché questa violerebbe il vincolo $SA \neq NT$ derivato dal problema originario. Possiamo quindi andare da quel nodo al nodo adiacente, con la variabile che possiamo chiamare $SANTQ$, e trovare che esiste una sola tupla, $(\text{rosso, verde, blu})$, che è consistente con la scelta per $SANTWA$. La stessa procedura viene ripetuta per i due nodi successivi, e in modo indipendente possiamo effettuare qualsiasi scelta per T .

Con RISOLUTORE-CSP-ALBERO possiamo risolvere qualsiasi problema di scomposizione ad albero in tempo $O(nd^2)$, in modo efficiente finché d rimane piccolo. Tornando al nostro esempio con 100 variabili booleane, se ogni nodo ha 10 variabili, allora $d = 2^{10}$ e dovremmo essere in grado di risolvere il problema in pochi secondi. Ma se esistesse un nodo con 30 variabili, servirebbero secoli.

Uno specifico grafo ammette diverse scomposizioni ad albero; la scelta deve cercare di rendere i sottoproblemi più piccoli possibile (notate che inserire tutte le variabili in un singolo nodo è tecnicamente un albero, ma non è utile). Data una scomposizione, la dimensione del nodo più grande meno uno è la **larghezza d’albero** (*tree width*) della scomposizione stessa; la larghezza d’albero di un grafo è definita come la larghezza minima tra tutte le sue scomposizioni ad albero. Se un grafo ha larghezza d’albero w , allora il problema può essere risolto in tempo $O(nd^{w+1})$ data la scomposizione ad albero corrispondente. Possiamo quindi dire che *i CSP i cui grafi dei vincoli hanno larghezza d’albero limitata sono risolvibili in tempo polinomiale*.

Sfortunatamente, trovare la scomposizione con larghezza d’albero minima è un problema NP-difficile, ma esistono metodi euristici che funzionano bene nella pratica. È meglio la scomposizione con insieme di taglio in tempo $O(d^c \cdot (n - c)d^2)$, oppure la scomposizione ad albero in tempo $O(nd^{w+1})$? Ogni volta che si ha un insieme di taglio dei cicli di dimensione c , esiste anche una larghezza d’albero di dimensione $w < c + 1$, a volte anche molto più piccola, perciò in riferimento al tempo si tende a favorire la scomposizione ad albero, ma il vantaggio dell’approccio con insieme di taglio dei cicli è che può essere eseguito in memoria lineare, mentre la scomposizione ad albero richiede memoria esponenziale in w .

larghezza d’albero



simmetria di valore

vincolo di rottura della simmetria

6.5.3 Simmetria di valore

Finora abbiamo considerato la struttura del grafo dei vincoli, ma può esserci un’importante struttura anche nei *valori* delle variabili, o nella struttura delle stesse relazioni di vincolo. Consideriamo il problema della colorazione della mappa con d colori. Per ogni soluzione consistente, esiste in effetti un insieme di $d!$ soluzioni formate permutando i nomi dei colori. Per esempio, sulla mappa dell’Australia sappiamo che WA, NT e SA devono avere colori diversi, ma ci sono $3! = 6$ modi per assegnare tre colori a tre regioni. In questo caso si parla di **simmetria di valore**. Ci piacerebbe ridurre lo spazio di ricerca di un fattore $d!$ rompendo la simmetria negli assegnamenti, e lo facciamo introducendo un **vincolo di rottura della simmetria**. Per il nostro esempio potremmo imporre un vincolo di ordinamento arbitrario, $NT < SA < WA$, che richieda che i tre valori siano disposti in ordine alfabetico. Questo vincolo garantisce che soltanto una delle $d!$ soluzioni è possibile: $\{NT = \text{blu}, SA = \text{rosso}, WA = \text{verde}\}$.

Per la colorazione della mappa è stato facile trovare un vincolo che eliminasse la simmetria, ma in generale è NP-difficile eliminare tutte le simmetrie. Tuttavia, la rottura della simmetria di valore è una tecnica che si è dimostrata importante ed efficace per un’ampia varietà di problemi.

6.6 Riepilogo

- I **problemi di soddisfacimento di vincoli** (spesso indicati con l'acronimo CSP) rappresentano uno stato con un insieme di coppie variabile/valore e rappresentano le condizioni per una soluzione con un insieme di vincoli sulle variabili. Molti importanti problemi del mondo reale possono essere espressi sotto forma di CSP.
- Molte tecniche di **inferenza** utilizzano i vincoli per escludere alcuni assegnamenti delle variabili. Tra queste vi sono consistenza di nodo, di arco, di cammino e k -consistenza.
- Per risolvere i CSP si usa comunemente la **ricerca con backtracking**, una forma particolare di ricerca in profondità. L'inferenza può essere inframmezzata con la ricerca.
- L'euristica **MRV** e quella **di grado** sono indipendenti dal dominio e possono essere usate durante la ricerca con backtracking per scegliere quale variabile considerare tra quelle rimanenti. Analogamente, l'euristica del **valore meno vincolante** aiuta a decidere quale valore provare per primo per una data variabile. Il backtracking si verifica quando non è più possibile assegnare un valore legale a una variabile. Il **backjumping guidato dai conflitti** torna direttamente all'origine del problema. L'**apprendimento dei vincoli** regista i conflitti che si incontrano durante la ricerca al fine di poterli evitare se si ripresentano più avanti.
- La ricerca locale con l'euristica **min-conflicts** è stata applicata con grande successo ai problemi di soddisfacimento di vincoli.
- La complessità del processo di risoluzione di un CSP è fortemente dipendente dalla struttura del suo grafo dei vincoli. I problemi strutturati ad albero possono essere risolti in tempo lineare. Il **condizionamento con insieme di taglio** può ridurre un CSP generico in uno strutturato ad albero, ed è piuttosto efficiente (richiede solo memoria lineare) se si riesce a trovare un insieme di taglio piccolo. Le tecniche di **scomposizione ad albero** trasformano il CSP in un albero di sottoproblemi e sono efficienti se la **larghezza d'albero** del grafo dei vincoli è ridotta; tuttavia, necessitano di memoria esponenziale nella larghezza d'albero del grafo di vincoli. Combinando condizionamento con insieme di taglio e scomposizione ad albero si può ottenere un miglior bilanciamento tra tempo e memoria.

Note storiche e bibliografiche

Il matematico greco Diofanto (c. 200–284) presentò e risolse diversi problemi con vincoli algebrici su equazioni, anche se non sviluppò un metodo generale. Oggi le equazioni con domini interi sono chiamate **equazioni diofantine**. Il matematico indiano Brahmagupta (c. 650) fu il primo a mostrare una soluzione generale per l'equazione $ax + by = c$ sul dominio degli interi. Metodi sistematici per la soluzione di equazioni lineari mediante eliminazione di variabili furono studiati da Gauss (1829); la soluzione dei vincoli di disegualanza lineare risalgono a Fourier (1827).

Anche i problemi di soddisfacimento di vincoli a dominio finito hanno una lunga storia. Per esempio, la **colorazione di grafi** (di cui la colorazione di mappe è un caso speciale) è un vecchio problema matematico.

La congettura dei quattro colori (che ipotizza che ogni grafo piano possa essere colorato con quattro colori o meno) fu formulata per la prima volta da Francis Guthrie, uno studente di De Morgan, nel 1852. Il problema resistette a ogni tentativo di soluzione (anche se ne furono pubblicate diverse) fino alla dimostrazione ottenuta da Appel e Haken (1977; cfr. il libro *Four Colors Suffice*, Wilson, 2004). I puristi manifestarono disappunto per il fatto che parte della dimostrazione utilizzasse un computer, perciò Georges Gonthier (2008), utilizzando il dimostratore di teoremi Coq, ricavò una dimostrazione formale che il programma di dimostrazione di Apple e Haken era corretto.

Per tutta la storia dell'informatica sono state considerate classi specifiche di problemi di soddisfaci-

mento di vincoli. Uno dei primi esempi più influenti fu il sistema SKETCHPAD (Sutherland, 1963), che risolveva vincoli geometrici su diagrammi e fu il precursore dei moderni programmi di disegno e CAD. L'identificazione dei CSP come classe *generale* è dovuta a Ugo Montanari (1974). La riduzione dei CSP di ordine superiore a CSP binari con l'aiuto di variabili ausiliarie (cfr. Esercizio 6.NARY) è stata effettuata per la prima volta nel XIX secolo dal logico Charles Sanders Peirce. Fu poi introdotta nella letteratura moderna dei CSP da Dechter (1990b) ed elaborata da Bacchus e van Beek (1998). I CSP che hanno preferenze nelle soluzioni sono studiati ampiamente nella letteratura dedicata all'ottimizzazione; cfr. Bistarelli *et al.* (1997) per una generalizzazione dell'infrastruttura CSP che permette di specificare preferenze.

I metodi di propagazione dei vincoli furono resi polari dal successo di Waltz (1975) sui problemi di etichettatura di linee di poliedri nel campo della visione artificiale. Waltz ha mostrato che, in molti problemi, la propagazione elimina completamente la necessità di effettuare backtracking. Montanari (1974) ha introdotto il concetto di grafi di vincoli e di propagazione mediante la consistenza dei cammini. Alan Mackworth (1977) ha proposto l'algoritmo AC-3 per assicurare la consistenza d'arco oltre all'idea generale di combinare il backtracking con la verifica di consistenza. AC-4, una versione più efficiente dell'algoritmo per la consistenza d'arco, fu sviluppato da Mohr e Henderson (1986), nel caso peggiore viene eseguito in tempo $O(cd^2)$, ma può essere più lento di AC-3 nel caso medio. L'algoritmo PC-2 (Mackworth, 1977) raggiunge la consistenza di cammino più meno nello stesso modo in cui AC-3 raggiunge la consistenza d'arco.

Subito dopo l'apparizione dell'articolo di Mackworth, i ricercatori hanno cominciato a sperimentare vari compromessi tra il costo delle verifiche di consistenza e i loro benefici in termini di riduzione dei tempi di ricerca. Haralick e Elliot (1980) hanno promosso l'algoritmo di verifica in avanti descritto da McGregor (1979), mentre Gaschnig (1979) ha suggerito di verificare completamente la consistenza d'arco dopo ogni assegnamento di variabile, un algoritmo che in seguito fu chiamato MAC da Sabin e Freuder (1994). Quest'ultimo lavoro fornisce prove abbastanza convincenti che, per i CSP più difficili, una completa verifica della consistenza d'arco risulta conveniente. Freuder (1978, 1982) ha investigato il concetto di k -consistenza e le sue relazioni con la complessità delle soluzioni dei CSP. Dechter e Dechter (1987) hanno introdotto la consistenza d'arco orientato. Apt (1999) descrive un'infra-

struttura algoritmica generica all'interno della quale si possono analizzare gli algoritmi di propagazione della consistenza; rassegne della letteratura sono state fornite da Bessière (2006) e Barták *et al.* (2010).

Metodi speciali per la gestione di vincoli di ordine superiore o globali sono stati sviluppati soprattutto nel contesto della **programmazione logica a vincoli**. Marriott e Stuckey (1998) offrono un'eccellente panoramica della ricerca in quest'area. Il vincolo *Tuttediverse* (*Alldiff*) è stato studiato da Regin (1994), Stergiou e Walsh (1999) e van Hoeve (2001). Esistono algoritmi di inferenza più complessi per *Tuttediverse* (cfr. van Hoeve e Katriel, 2006) che propagano più vincoli ma sono più costosi in termini computazionali. I vincoli sugli estremi furono incorporati nella programmazione logica a vincoli da Van Hentenryck *et al.* (1998). Una rassegna sui vincoli globali è fornita da van Hoeve e Katriel (2006).

Il Sudoku è diventato il CSP più noto ed è stato descritto come tale da Simonis (2005). Agerbeck e Hansen (2008) descrivono alcune delle strategie e mostrano che il Sudoku su una griglia $n^2 \times n^2$ rientra nella classe dei problemi NP-difficili.

Nel 1850, C. F. Gauss descrisse un algoritmo ricorsivo con backtracking per risolvere il problema delle 8 regine, che era stato pubblicato nella rivista tedesca di scacchi *Schachzeitung* nel 1848. Gauss chiamò il suo metodo *Tattonniren*, dalla parola francese *tâtonner* che significa andare a tentoni, brancolare nel buio.

Secondo Donald Knuth (comunicazione personale), R. J. Walker introdusse il termine *backtrack* negli anni 1950. Walker (1960) descrisse l'algoritmo di base per il backtracking e lo utilizzò per trovare tutte le soluzioni del problema delle 13 regine. Golomb e Baumert (1965) formularono, con esempi, la classe generale dei problemi combinatori a cui si può applicare il backtracking e introdussero quella che ora chiamiamo l'euristica MRV. Bitner e Reingold (1975) presentarono un'importante panoramica della letteratura sulle tecniche di backtracking. Brelaz (1979) usò l'euristica di grado per rompere i pareggi dopo l'applicazione dell'euristica MRV. L'algoritmo risultante, in tutta la sua semplicità, rimane il metodo migliore per k -colorare grafi arbitrari. Haralick ed Elliott (1980) proposero l'euristica del valore meno vincolante.

Il metodo base del backjumping è dovuto a John Gaschnig (1977, 1979). Kondrak e van Beek (1997) hanno mostrato che quest'algoritmo è sostanzialmente generalizzato dalla verifica in avanti. Il backjumping guidato dai conflitti fu inventato da Prosser (1993). Dechter (1990a) introdusse il backjumping ba-

sato su grafo, che limita la complessità degli algoritmi basati sul backjumping in funzione del grafo dei vincoli (Dechter e Frost, 2002).

Una forma molto generale di backtracking intelligente fu inizialmente sviluppata da Stallman e Sussman (1977): la loro tecnica, il **backtracking guidato dalle dipendenze**, combina backjumping e apprendimento di no-good (McAllester, 1990) e ha portato allo sviluppo dei **sistemi di mantenimento della verità** (Doyle, 1979), che discuteremo nel Paragrafo 10.6.2. Il collegamento tra le due aree è analizzato da de Kleer (1989).

Il lavoro di Stallman e Sussman ha anche introdotto l'idea dell'**apprendimento dei vincoli**, in cui i risultati parziali della ricerca possono essere salvati e riutilizzati più tardi. L'idea fu formalizzata da Dechter (1990a). Il **backmarking** (Gaschnig, 1979) è un metodo particolarmente semplice basato sulla memorizzazione di coppie di assegnamenti consistenti e inconsistenti per evitare di controllare nuovamente i vincoli. Il backmarking può essere combinato con il backjumping guidato dai conflitti; Kondrak e van Beek (1997) presentano un algoritmo ibrido che generalizza ognuno dei due metodi presi separatamente.

Il metodo del **backtracking dinamico** (Ginsberg, 1993) conserva gli assegnamenti parziali di successo per impiegarli successivamente in caso di backtracking e di nuovi assegnamenti per lo stesso sottoinsieme di variabili. Moskewicz *et al.* (2001) mostrano come usare queste tecniche e altre per creare un efficiente risolutore SAT. Studi empirici di diversi metodi con backtracking randomizzati sono stati effettuati da Gomes *et al.* (2000) e Gomes e Selman (2001). Van Beek (2006) presenta una rassegna sul backtracking.

La ricerca locale nei problemi di soddisfacimento di vincoli è stata resa popolare dal lavoro di Kirkpatrick *et al.* (1983) sul simulated annealing (cfr. Capitolo 4), ampiamente usato nei problemi di configurazione VLSI e di scheduling. Beck *et al.* (2011) forniscono una panoramica sui lavori recenti in tema di job-shop scheduling. L'euristica min-conflicts fu proposta per la prima volta da Gu (1989) e sviluppata indipendentemente da Minton *et al.* (1992). Sosic e Gu (1994) hanno mostrato come poteva essere applicata per risolvere il problema con 3.000.000 di regine in meno di un minuto. L'incredibile successo della ricerca locale con euristica min-conflicts sul problema delle n regine portò a una riconsiderazione della natura e della prevalenza dei problemi "facili" e "difficili". Peter Cheeseman *et al.* (1991) hanno studiato la difficoltà dei CSP generati casualmente e hanno scoperto che quasi tutti i problemi così creati sono banal-

mente facili oppure non hanno soluzioni: per trovare istanze di problemi "difficili" si devono configurare i parametri del generatore di problemi in un determinato, piccolo intervallo, all'interno del quale è risolvibile circa la metà dei problemi. Discuteremo ulteriormente questo fenomeno nel Capitolo 7.

Konolige (1994) mostrò che la ricerca locale è inferiore alla ricerca con backtracking in problemi con un certo grado di struttura locale; questo ha portato a lavori che hanno combinato ricerca locale e inferenza, come quello di Pinkas e Dechter (1995). Hoos e Tsang (2006) presentano una rassegna sulle tecniche di ricerca locale, argomento dei libri di testo di Hoos e Stützle (2004) e Aarts e Lenstra (2003).

Gli studi sulla struttura e la complessità dei CSP hanno avuto origine con Freuder (1985) e Mackworth e Freuder (1985), i quali hanno dimostrato che la ricerca su alberi arco-consistenti non ha bisogno di backtracking. Un risultato simile, esteso agli ipergrafo aciclici, fu sviluppato nella comunità dei database (Beeri *et al.*, 1983). Bayardo e Miranker (1994) presentano un algoritmo per CSP con struttura ad albero che viene eseguito in tempo lineare senza alcuna pre-laborazione. Dechter (1990a) descrive l'approccio dell'insieme di taglio dei cicli.

Dalla pubblicazione di questi articoli sono stati fatti molti progressi e ottenuta una comprensione più generale sulla correlazione tra la complessità della risoluzione di un CSP e la struttura del suo grafo dei vincoli. Il concetto di larghezza d'albero fu introdotto dai teorici dei grafi Robertson e Seymour (1986). Dechter e Pearl (1987, 1989), partendo dal lavoro di Freuder, applicarono un concetto correlato (che chiamarono **larghezza indotta**, ma è identico alla larghezza d'albero) ai problemi di soddisfacimento di vincoli, sviluppando la tecnica di scomposizione ad albero che abbiamo descritto nel Paragrafo 6.5.

Partendo da questo lavoro, insieme ad alcuni risultati della teoria dei database, Gottlob *et al.* (1999a, 1999b) svilupparono il concetto di **larghezza di iperalbero**, basato sulla descrizione del CSP come ipergrafo. Oltre a dimostrare che ogni CSP con larghezza di iperalbero w può essere risolto in un tempo $O(n^{w+1} \log n)$, hanno anche mostrato che la larghezza di iperalbero è più generale di tutte le misure di "larghezza" precedentemente formulate, nel senso che ci sono casi in cui la larghezza di iperalbero è limitata e le altre misure no.

L'algoritmo RELSAT di Bayardo e Schrag (1997) combinava apprendimento dei vincoli e backjumping e superò le prestazioni di molti altri algoritmi dell'epoca. Questo portò ad algoritmi di ricerca AND/OR ap-

plicabili sia ai CSP, sia al ragionamento probabilistico (Dechter e Mateescu, 2007). Brown *et al.* (1988) introducono l’idea di violazione della simmetria nei CSP, e Gent *et al.* (2006) forniscono una rassegna.

Il campo del **soddisfacimento di vincoli distribuiti** esamina la risoluzione di CSP in cui vi è una collezione di agenti, ognuno dei quali controlla un sottoinsieme delle variabili vincolate. A partire dal 2000 si sono tenuti workshop annuali sul problema, che ha visto una buona copertura anche altrove (Collin *et al.*, 1999; Pearce *et al.*, 2008).

Confrontare algoritmi per CSP è per lo più una scienza empirica: pochi risultati teorici mostrano che un algoritmo prevale nettamente su un altro per tutti i problemi; dobbiamo invece eseguire esperimenti per vedere quali algoritmi hanno prestazioni migliori su istanze tipiche dei problemi. Come sottolinea Hooker (1995), dobbiamo prestare attenzione a distinguere tra test competitivi (come quelli utilizzati nelle competizioni tra algoritmi basate sul tempo di esecuzione) e test scientifici, il cui obiettivo è quello di individuare

le proprietà di un algoritmo che ne determinano l’efficacia su una classe di problemi.

I testi di Apt (2003), Dechter (2003), Tsang (1993) e Lecoutre (2009), oltre alla raccolta di Rossi *et al.* (2006) sono eccellenti risorse sull’elaborazione dei vincoli. Esistono molte buone rassegne, tra cui quelle di Dechter e Frost (2002) e Bartak *et al.* (2001). Carbonnel e Cooper (2016) hanno presentato una rassegna delle classi trattabili di CSP. Kondrak e van Beek (1997) hanno svolto uno studio analitico degli algoritmi di ricerca con backtracking, mentre il lavoro di Bacchus e van Run (1995) ha un taglio più empirico. La programmazione a vincoli è trattata nei libri di Apt (2003) e Fruhwirth e Abdennadher (2003). Articoli sul soddisfacimento di vincoli compaiono regolarmente su *Artificial Intelligence* e sulla rivista specializzata *Constraints*. I più recenti risolutori SAT sono descritti nell’annuale International SAT Competition. Il più importante congresso è l’International Conference on Principles and Practice of Constraint Programming, spesso indicato brevemente con *CP*.

P A R T E

3

Conoscenza, ragionamento e pianificazione

Capitolo 7 Agenti logici

Capitolo 8 Logica del primo ordine

**Capitolo 9 Inferenza nella logica del primo
ordine**

**Capitolo 10 Rappresentazione
della conoscenza**



CAPITOLO

7

- 7.1 Agenti basati sulla conoscenza
- 7.2 Il mondo del wumpus
- 7.3 Logica
- 7.4 Logica proposizionale: una logica molto semplice
- 7.5 Dimostrazione di teoremi nella logica proposizionale
- 7.6 Model checking proposizionale efficiente
- 7.7 Agenti basati sulla logica proposizionale
- 7.8 Riepilogo
Note storiche e bibliografiche

Agenti logici

Nel quale progettiamo agenti che possono costruire rappresentazioni di mondi complessi, applicare processi di inferenza per derivare nuove rappresentazioni e usarle per dedurre cosa fare.

Gli esseri umani apparentemente sanno delle cose, e ciò che sanno li aiuta a fare. Nell'IA, gli **agenti basati sulla conoscenza** decidono quali azioni intraprendere utilizzando un processo di **ragionamento** che opera su una **rappresentazione** interna di conoscenza.

Gli agenti risolutori di problemi dei Capitoli 3 e 4 conoscono cose, ma solo in un senso molto limitato e non flessibile. Sanno quali azioni sono disponibili e quale sarà il risultato dell'esecuzione di un'azione specifica da uno stato specifico, ma non conoscono fatti generali. Un agente per la ricerca di un itinerario stradale non sa che la lunghezza di una strada non può essere un numero negativo di chilometri. Un agente per un rompicapo a 8 tasselli non sa che due tasselli non possono occupare lo stesso spazio. La conoscenza di cui dispongono questi agenti è molto utile per trovare un cammino che conduca dall'inizio a un obiettivo, ma non per altri scopi.

Le rappresentazioni atomiche utilizzate dagli agenti risolutori di problemi sono molto limitanti. In un ambiente parzialmente osservabile, per esempio, l'unica scelta a disposizione di un agente per rappresentare ciò che sa circa lo stato corrente è quella di elencare tutti i possibili stati concreti. Potremmo indicare a un essere umano l'obiettivo di raggiungere in auto una città statunitense con meno di 10.000 abitanti, ma per indicare la stessa cosa a un agente risolutore di problemi dovremmo descrivere l'obiettivo come insieme esplicito delle circa 16.000 città statunitensi che soddisfano il requisito di popolazione.

Nel Capitolo 6 abbiamo introdotto la prima rappresentazione fattorizzata, in cui gli stati sono rappresentati come assegnamenti di valori a variabili; questo è un passo nella direzione giusta, che consente ad alcune parti dell’agente di lavorare in modo indipendente dal dominio e permette di ottenere algoritmi più efficienti. In questo capitolo porteremo questo passo alla sua conclusione logica, per così dire: svilupperemo la **logica** come classe generale di rappresentazioni per supportare agenti basati sulla conoscenza. Tali agenti possono combinare e ricombinare informazioni per una miriade di scopi. Questo processo può essere molto distante dalle esigenze del momento, come avviene quando un matematico dimostra un teorema o un astronomo calcola l’aspettativa di vita del pianeta terra. Gli agenti basati sulla conoscenza possono intraprendere nuove attività espresse sotto forma di obiettivi descritti esplicitamente, possono ottenere rapidamente nuove competenze ricevendo o apprendendo conoscenze aggiuntive e possono adattarsi ai cambiamenti dell’ambiente modificando e aggiornando la conoscenza rilevante.

Cominceremo con il presentare la struttura generale dell’agente nel Paragrafo 7.1. Il Paragrafo 7.2 introduce un nuovo, semplice ambiente, il mondo del wumpus, e descrive il funzionamento di un agente basato sulla conoscenza senza alcun dettaglio tecnico. Poi spiegheremo i principi generali della **logica** nel Paragrafo 7.3 e i dettagli della **logica proposizionale** nel Paragrafo 7.4. La logica proposizionale è una rappresentazione fattorizzata; benché sia meno espressiva della **logica del primo ordine** (Capitolo 8), che corrisponde alla canonica rappresentazione strutturata, illustra tutti i concetti fondamentali della logica. Inoltre, la logica proposizionale è supportata da tecnologie per l’inferenza ben sviluppate, che presentiamo nei Paragrafi 7.5 e 7.6. Infine, il Paragrafo 7.7 unisce il concetto di agente basato sulla conoscenza con la tecnologia della logica proposizionale per costruire alcuni semplici agenti per il mondo del wumpus.

7.1 Agenti basati sulla conoscenza

base di conoscenza

formula

**linguaggio
di rappresentazione
della conoscenza**

assioma

inferenza

conoscenza iniziale

Il componente più importante degli agenti basati sulla conoscenza è appunto la **base di conoscenza**, o KB (dall’inglese *knowledge base*). Informalmente, la base di conoscenza è costituita da un insieme di **formule**, espresse mediante un **linguaggio di rappresentazione della conoscenza**. Ogni formula rappresenta un’asserzione sul mondo. Quando una formula è data per buona senza essere ricavata da altre formule, la chiamiamo **assioma**.

La base di conoscenza deve prevedere meccanismi per aggiungere nuove formule e per le interrogazioni (o *query*). I nomi standard per queste due azioni sono rispettivamente TELL (asserisci) e ASK (chiedi): entrambe possono comportare un processo di **inferenza**, ovvero la derivazione di nuove formule a partire da quelle conosciute. L’inferenza deve soddisfare il requisito fondamentale che la risposta a ogni richiesta (ASK) posta alla base di conoscenza sia una conseguenza di quello che le è stato detto (attraverso TELL) in precedenza. Più avanti nel capitolo saremo più precisi riguardo alla fondamentale parola “conseguenza”. Per adesso, accontentiamoci di dire che il processo di inferenza non può semplicemente inventarsi i fatti.

La Figura 7.1 mostra lo schema di un programma agente basato sulla conoscenza: come tutti i nostri agenti, prende in input una percezione e restituisce un’azione. L’agente mantiene in memoria una base di conoscenza, *KB*, che può contenere **conoscenza iniziale** (*background knowledge*).

Ogni volta che viene invocato, il programma agente fa tre cose: prima di tutto comunica le sue percezioni (attraverso TELL) alla base di conoscenza. Quindi le chiede (ASK) quale azione eseguire. Rispondere a questa domanda può comportare un lungo processo di ragionamento sullo stato corrente del mondo, le conseguenze delle possibili azioni e così via. Una volta che è stata scelta un’azione l’agente la registra con TELL nella base di conoscenza e la restituisce in modo che possa essere eseguita.

```

function AGENTE-KB(percezione) returns un'azione
  persistent: KB, una base di conoscenza
    t, un contatore, inizializzato a 0, che indica il tempo

  TELL(KB, COSTRUISCI-FORMULA-PERCEZIONE(percezione, t))
  azione  $\leftarrow$  ASK(KB, COSTRUISCI-INTERROGAZIONE-AZIONE(t))
  TELL(KB, COSTRUISCI-FORMULA-AZIONE(azione, t))
  t  $\leftarrow$  t + 1
  return azione

```

Figura 7.1 Un generico agente basato sulla conoscenza. Data una percezione, l'agente la aggiunge alla propria base di conoscenza, chiede alla base di conoscenza quale sia l'azione migliore e dice alla base di conoscenza che ha in effetti intrapreso tale azione.

I dettagli del linguaggio di rappresentazione sono racchiusi nelle tre funzioni che implementano l'interfaccia tra i sensori e attuatori da una parte e il sistema interno di rappresentazione e ragionamento dall'altra. COSTRUISCI-FORMULA-PERCEZIONE costruisce una formula che asserisce che un dato istante temporale l'agente ha ricevuto una data percezione. COSTRUISCI-INTERROGAZIONE-AZIONE costruisce una formula che chiede quale azione intraprendere in quel momento. Infine COSTRUISCI-FORMULA-AZIONE costruisce una formula che asserisce che l'azione scelta è stata eseguita. I dettagli del meccanismo di inferenza sono nascosti dentro TELL e ASK: li riveleremo nei prossimi paragrafi.

L'agente della Figura 7.1 ha un aspetto molto simile a quelli dotati di uno stato interno che abbiamo descritto nel Capitolo 2. Tuttavia le limitazioni intrinseche di TELL e ASK fanno sì che l'agente basato sulla conoscenza non sia un programma arbitrario per il calcolo di azioni. Un agente siffatto si presta bene a essere descritto a **livello della conoscenza**, in cui per determinare il comportamento basta specificare solamente ciò che l'agente conosce e i suoi obiettivi. Per esempio, un taxi automatico potrebbe avere l'obiettivo di portare un passeggero da San Francisco a Marin County, e potrebbe sapere che il Golden Gate Bridge è l'unico collegamento tra le due località. Allora ci aspetteremo che il taxi attraversi il ponte, *perché sa che così facendo raggiungerà il suo obiettivo*. Notate che quest'analisi è del tutto indipendente dal funzionamento del taxi a **livello dell'implementazione**: non importa se la sua conoscenza della geografia è realizzata con liste dinamiche o mappe di pixel, o se per ragionare manipola stringhe di simboli memorizzate in registri o propaga segnali rumorosi in una rete neurale.

Un agente basato sulla conoscenza può essere costruito semplicemente dicendogli (TELL) ciò che deve sapere. Iniziando con una base di conoscenza vuota, il progettista dell'agente può utilizzare TELL su varie formule, passandole all'agente una per una, finché l'agente sa come operare nel proprio ambiente. Questo procedimento prende il nome di approccio **dichiarativo** alla realizzazione di sistemi. In contrapposizione a questo, l'approccio **procedurale** codifica direttamente in un programma i comportamenti desiderati sotto forma di codice. Negli anni 1970 e 1980 ci sono stati dibattiti molto accesi tra i difensori dei due approcci: oggi l'opinione comune è che un agente di successo debba combinare sia elementi dichiarativi che procedurali, e che la conoscenza dichiarativa spesso possa essere compilata in codice procedurale più efficiente.

Possiamo anche fornire a un agente basato sulla conoscenza dei meccanismi che gli permettano di apprendere. Questi meccanismi, che discuteremo nel Capitolo 19 del Volume 2, partono da una serie di percezioni per dare origine a nuova conoscenza generale riguardante l'ambiente. Un agente in grado di apprendere può essere completamente autonomo.

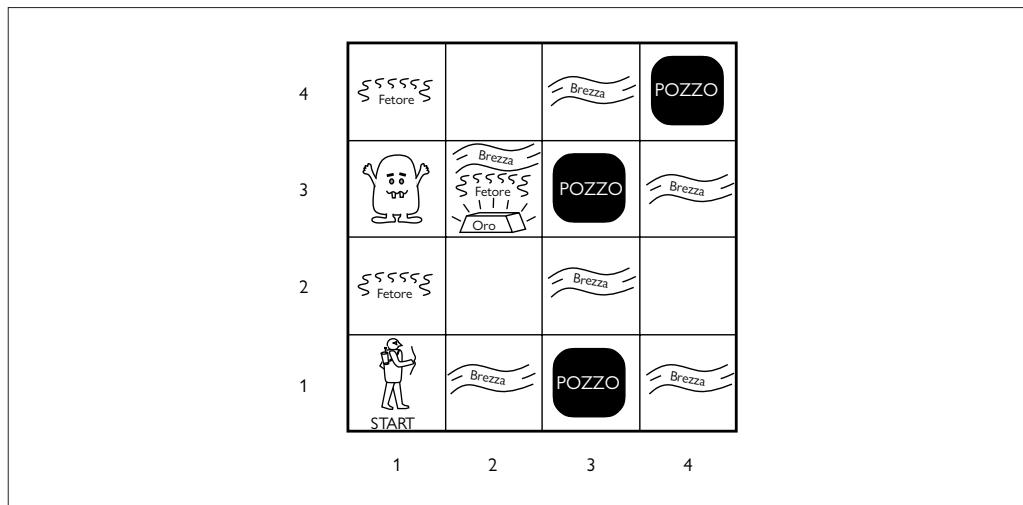
**livello
della conoscenza**

**livello
della
implementazione**

**dichiarativo
procedurale**

Figura 7.2

Un tipico mondo del wumpus. L'agente si trova nell'angolo in basso a sinistra, rivolto a est (verso destra).



7.2 Il mondo del wumpus

mondo del wumpus

In questo paragrafo descriviamo un ambiente in cui gli agenti basati sulla conoscenza possono dimostrare tutto il loro valore. Il **mondo del wumpus** è una caverna formata da stanze collegate da passaggi. Acquattato da qualche parte nell'ombra c'è il terribile wumpus, una bestia pronta a divorare chiunque entri. Il wumpus può essere colpito da lontano da un agente, che però ha una sola freccia. Alcune stanze contengono pozzi senza fondo che intrappoleranno chiunque entri (tranne il wumpus, che è troppo grande per caderci dentro). L'unica caratteristica positiva di quest'ambiente è la possibilità di trovare un mucchio d'oro. Benché risulti piuttosto blando se confrontato con i videogiochi più recenti, il mondo del wumpus mette in luce alcuni importanti aspetti relativi all'intelligenza.

Un esempio di mondo del wumpus è mostrato nella Figura 7.2. La definizione precisa dell'ambiente è fornita, come suggerito nel Paragrafo 2.3, sotto forma di descrizione PEAS.

- **Misura di prestazione:** +1000 se si riesce a uscire dalla caverna portando l'oro, -1000 se si cade in un pozzo o si viene divorati dal wumpus, -1 per ogni azione eseguita e -10 per l'uso della freccia. Il gioco termina quando l'agente muore o quando esce dalla caverna.
- **Ambiente:** una griglia 4×4 di stanze, circondata da pareti di delimitazione. L'agente comincia sempre nella posizione etichettata [1,1], rivolto verso est. Le posizioni dell'oro e del wumpus sono scelte casualmente, con una distribuzione uniforme, tra tutti i riquadri tranne quello iniziale. Inoltre, tutti i riquadri tranne quello iniziale hanno una probabilità 0,2 di contenere un pozzo senza fondo.
- **Attuatori:** l'agente può muoversi *Avanti*, *GiraSinistra* di 90° o *GiraDestra* di 90° . L'agente muore miserevolmente se entra in un riquadro che contiene un pozzo o il wumpus vivo: entrare in una stanza con un wumpus morto non è rischioso, anche se il fetore è quasi insopportabile. Se un agente tenta di muoversi in avanti e si scontra con un muro, non si muove. L'azione *Afferra* può essere usata per prendere l'oro che si trova nella stessa stanza. L'azione *Scocca* scoglia una freccia in linea retta nella direzione verso cui l'agente è rivolto; la freccia continua la sua corsa finché non colpisce il wumpus uccidendolo o sbatte contro un muro. L'agente possiede una sola freccia, per cui l'unica azione *Scocca* ad avere effetto è la prima. Infine, l'azione *Esci* può essere utilizzata per arrampicarsi fuori dalla caverna, ma soltanto dalla stanza [1,1].
- **Sensori:** l'agente ha cinque sensori, ognuno dei quali fornisce un singolo bit di informazione:

- nei riquadri direttamente adiacenti (non in diagonale) a quello che contiene il wumpus, l’agente percepirà un *Fetore*¹;
- nei riquadri direttamente adiacenti a un pozzo l’agente percepirà una *Brezza*;
- nel riquadro che contiene l’oro l’agente percepirà uno *Scintillio*;
- qualora l’agente dovesse sbattere contro un muro, percepirà un *Urto*;
- quando il wumpus viene ucciso emette un *Ululato* straziante, percepibile nell’intera caverna.

Le percezioni saranno fornite all’agente sotto forma di una lista di cinque simboli: per esempio, se c’è puzza e movimento d’aria ma nessun scintillio, urto o ululato, l’agente riceverà la percezione [*Fetore*, *Brezza*, *None*, *None*, *None*].

Possiamo caratterizzare l’ambiente del wumpus sulla base delle varie caratteristiche discusse nel Capitolo 2. È chiaramente deterministico, discreto, statico e monoagente (il wumpus non si muove, fortunatamente). È sequenziale, perché si può raggiungere un premio soltanto dopo aver intrapreso molte azioni. È parzialmente osservabile, perché alcuni aspetti dello stato non sono direttamente percepibili: la posizione dell’agente, lo stato di salute del wumpus e la disponibilità di una freccia. Per quanto riguarda le posizioni dei pozzi e del wumpus, potremmo considerarle quali parti inosservate dello stato, nel qual caso il modello di transizione per l’ambiente sarebbe del tutto noto, e trovando le posizioni dei pozzi si completerebbe la conoscenza dello stato da parte dell’agente. Oppure potremmo dire che il modello di transizione è sconosciuto perché l’agente non sa quali azioni *Avanti* sono fatali, nel qual caso, scoprendo le posizioni di pozzi e wumpus si completerebbe la conoscenza del modello di transizione da parte dell’agente.

Per un agente nell’ambiente, la difficoltà principale è legata alla sua iniziale ignoranza della configurazione dell’ambiente; per superarla sembra essere necessario un ragionamento logico. Nella maggior parte delle istanze del mondo del wumpus, l’agente può recuperare l’oro in sicurezza. Occasionalmente, dovrà scegliere se tornare a casa a mani vuote o rischiare la vita per ottenerlo. Il 21% circa degli ambienti è decisamente antisportivo, perché l’oro si trova in fondo a un pozzo o in una stanza circondata da pozzi.

Esaminiamo il comportamento di un agente basato sulla conoscenza che esplora l’ambiente mostrato nella Figura 7.2. Utilizziamo un linguaggio di rappresentazione della conoscenza informale che consiste nello scrivere simboli in una griglia (come nelle Figure 7.3 e 7.4).

La base di conoscenza iniziale dell’agente contiene le regole del mondo, come le abbiamo descritte in precedenza: in particolare, l’agente sa di essere nella posizione [1,1] e che la stanza è sicura; indichiamo ciò rispettivamente con “A” e “OK” nella stanza [1,1].

La prima percezione è [*None*, *None*, *None*, *None*, *None*], da cui l’agente può dedurre che le stanze adiacenti, [1,2] e [2,1], sono sicure, quindi OK. La Figura 7.3(a) mostra lo stato della conoscenza dell’agente a questo punto.

Un agente cauto entrerà solo nelle stanze segnate con OK. Supponiamo che l’agente decida di avanzare in [2,1]. L’agente percepisce un movimento d’aria o brezza (denotata da “B”) in [2,1], perciò in una stanza adiacente dev’esserci un pozzo. Questo non può essere in [1,1], per cui deve trovarsi in [2,2] o [3,1], o anche in entrambe le posizioni. La notazione “P?” nella Figura 7.3(b) indica questa possibilità. A questo punto, c’è un solo riquadro marcato con OK e non ancora visitato; il nostro prudente agente tornerà quindi indietro in [1,1] e da lì procederà in [1,2].

L’agente percepisce un fetore in [1,2], che dà come risultato lo stato di conoscenza della Figura 7.4(a). Il fetore nella stanza [1,2] significa che il wumpus è vicino. Ma il mostro non

¹ Presumibilmente anche il riquadro contenente il wumpus ha un fetore, ma ogni agente che entra in tale riquadro viene mangiato prima di poter percepire alcunché.

(a)

1,4	2,4	3,4	4,4	
1,3	2,3	3,3	4,3	
1,2	2,2	3,2	4,2	
OK				
1,1	A OK	2,1 OK	3,1	4,1

(b)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	P? OK	4,2
1,1	V OK	2,1 A B OK	3,1 P? 4,1

Figura 7.3 Il primo passo dell'agente nel mondo del wumpus. (a) La situazione iniziale, dopo la percezione [None, None, None, None, None]. (b) Dopo la mossa in [2, 1] e con la percezione [None, Brezza, None, None, None].

(a)

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P! OK	4,1

(b)

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A F O B	P? 4,3	
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	P! 4,1	

Figura 7.4 Due fasi successive dell'esplorazione dell'agente. (a) Dopo la mossa in [1, 1] e poi in [1, 2], con la percezione [Fetore, None, None, None, None]. (b) Dopo la mossa in [2, 2] e poi in [2, 3], con la percezione [Fetore, Brezza, Scintillio, None, None].

può trovarsi in [1,1] per definizione, e neppure in [2,2], altrimenti l'agente avrebbe percepito la puzza anche nella stanza [2,1]. L'agente può quindi dedurre che il wumpus si trova in [1,3], ciò che indichiamo con la notazione W!. Inoltre, la mancanza di *Brezza* in [1,2] implica che non ci siano pozzi in [2,2], e a questo punto si può inferire con certezza che ce ne dev'essere uno nella posizione [3,1]. Quest'ultima deduzione è abbastanza difficile, perché si fonda su conoscenze apprese in diversi momenti e in posti differenti; inoltre si basa sull'assenza di una particolare percezione per compiere un passo fondamentale nel ragionamento.

Ora l'agente si è convinto oltre ogni dubbio che in [2,2] non ci sono pozzi né mostri, per cui è OK andarci. Non mostreremo lo stato della conoscenza dell'agente in [2,2]; ci limiteremo a supporre che da lì si giri verso sinistra ed entri nella stanza [2,3], raggiungendo la si-

tuazione della Figura 7.4(b). In [2,3] l’agente percepisce lo scintillio dell’oro, così può raccoglierlo e tornare a casa.

Noteate che ogni volta che l’agente trae una conclusione dalle informazioni disponibili, tale conclusione *sarà sempre corretta* a patto che lo sia l’informazione di partenza. Questa è una proprietà fondamentale del ragionamento logico. Nel resto del capitolo spiegheremo come costruire agenti logici che possono rappresentare l’informazione necessaria e trarre conclusioni come quelle che abbiamo descritto nei paragrafi precedenti.

7.3 Logica

Questo paragrafo introduce i concetti fondamentali della rappresentazione e del ragionamento logico. Questi concetti sono indipendenti da qualsiasi forma particolare di logica, perciò rimanderemo al prossimo paragrafo i dettagli tecnici legati a tali forme, utilizzando invece il familiare esempio della semplice aritmetica.

Nel Paragrafo 7.1 abbiamo detto che le basi di conoscenza sono costituite da formule. Queste formule sono espresse secondo le regole della **sintassi** del linguaggio di rappresentazione, che specifica quali di esse sono “ben formate”. La nozione di sintassi è abbastanza intuitiva quando si parla dell’aritmetica a noi familiare: “ $x + y = 4$ ” è una formula ben formata, mentre “ $x4y+ =$ ” non lo è.

Una logica deve anche definire la **semantica**, o il significato delle formule. La semantica definisce la **verità** delle formule rispetto a ogni **mondo possibile**. Per esempio, la semantica normalmente adottata per l’aritmetica specifica che la formula “ $x + y = 4$ ” è vera in un mondo in cui x vale 2 e così y , ma falsa in un mondo in cui entrambe le variabili valgono 1. Nelle logiche standard, ogni formula dev’essere o vera o falsa in ogni mondo possibile: non esistono possibilità intermedie.²

Quando dovremo essere precisi, al posto di “mondo possibile” useremo il termine **modello**. Laddove i mondi possibili possono essere considerati ambienti (potenzialmente) reali in cui un agente potrebbe o non potrebbe trovarsi, i modelli sono astrazioni matematiche, e ognuno di essi ha un valore di verità fissato (vero o falso) per ogni formula. Informalmente, potremmo per esempio pensare a un mondo possibile in cui vi siano x uomini e y donne seduti intorno a un tavolo per giocare a bridge, e la formula $x + y = 4$ sia vera quando vi sono quattro persone in totale. Formalmente, i modelli possibili non sono altro che tutti i modi in cui si possono assegnare valori interi e non negativi alle variabili x e y . Ogni assegnamento determina il valore di verità di qualsiasi formula aritmetica su tali variabili. Se una formula α è vera in un modello m , diciamo che m **soddisfa** α o talvolta che m è un **modello di** α . Utilizziamo la notazione $M(\alpha)$ per indicare l’insieme di tutti i modelli di α .

Ora che abbiamo definito la nozione di verità possiamo parlare del ragionamento logico. Per questo dobbiamo introdurre la relazione di **conseguenza logica** tra formule, che significa che una *segue logicamente* dall’altra. In notazione matematica, si scrive:

$$\alpha \models \beta$$

per indicare che da α segue logicamente β . La definizione formale di conseguenza logica è la seguente: $\alpha \models \beta$ se e solo se, in ogni modello in cui α è vera, anche β lo è. Utilizzando la notazione appena introdotta, possiamo scrivere:

$$\alpha \models \beta \text{ se e solo se } M(\alpha) \subseteq M(\beta)$$

(Notate l’orientamento del simbolo \subseteq : se $\alpha \models \beta$, allora α è un’asserzione più forte di β , perché esclude un *maggior numero* di mondi possibili). La relazione di conseguenza logica è nota a chiunque conosca l’aritmetica; sappiamo tutti che dalla formula $x = 0$ segue la for-

sintassi

semantica

verità

mondo possibile

modello

soddisfazione

conseguenza logica

² La **logica fuzzy** o sfumata, che presenteremo nel Capitolo 13, permette di esprimere gradi di verità.

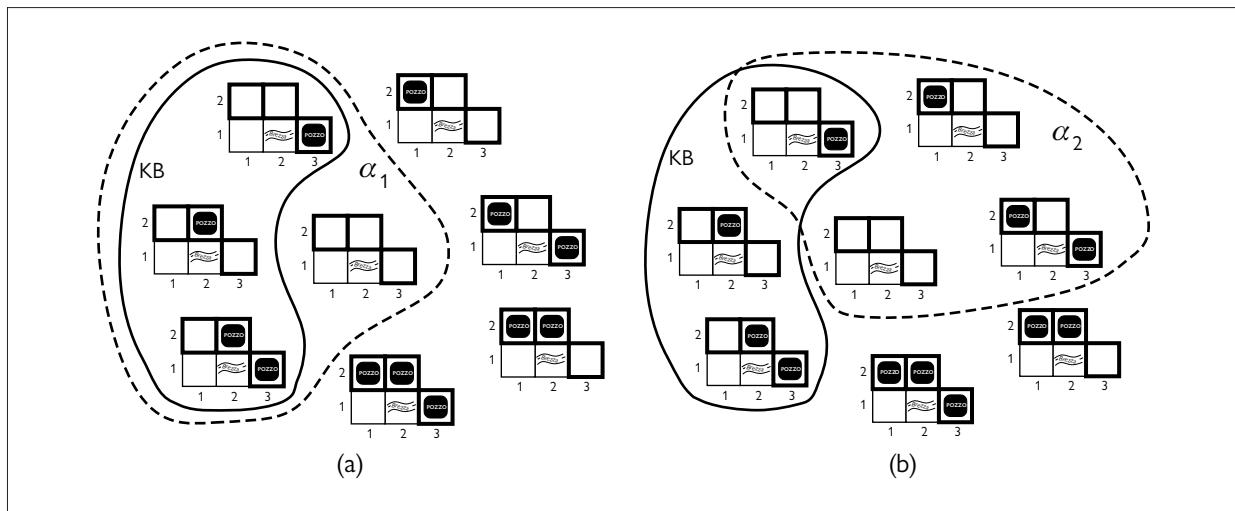


Figura 7.5 I possibili modelli per la presenza di pozzi nelle stanze [1,2], [2,2] e [3,1]. La KB corrispondente al non aver rilevato nulla in [1,1] e una brezza in [2,1] è mostrato dalla linea continua. (a) La linea tratteggiata mostra modelli di α_1 (nessun pozzo in [1,2]). (b) La linea tratteggiata mostra modelli di α_2 (nessun pozzo in [2,2]).

mula $xy = 0$. È infatti palese che in ogni modello in cui x è zero, anche xy è zero, indipendentemente dal valore di y .

Possiamo applicare lo stesso tipo di analisi all'esempio di ragionamento nel mondo del wumpus che abbiamo presentato nel paragrafo precedente. Considerate la situazione nella Figura 7.3(b): l'agente non ha percepito nulla in [1,1] e un movimento d'aria in [2,1]. Queste percezioni, unite alla conoscenza da parte dell'agente delle regole del mondo del wumpus, costituiscono la KB. L'agente è interessato a sapere se le stanze adiacenti [1,2], [2,2] e [3,1] contengono pozzi. Ognuno dei riquadri potrebbe contenere un pozzo oppure no, per cui (ignorando per ora gli altri aspetti del mondo) ci sono $2^3 = 8$ modelli possibili, che abbiamo riportato nella Figura 7.5.³

La KB può essere vista come un insieme di formule o come una singola formula che assicura tutte le formule individuali. La KB è falsa nei modelli che contraddicono le conoscenze dell'agente: per esempio in qualsiasi modello in cui [1,2] contiene un pozzo, dato che l'agente non ha percepito alcun movimento d'aria in [1,1]. In effetti, i modelli in cui la KB è vera sono solo tre, delimitati con una linea continua nella Figura 7.5. Ora consideriamo le possibili conclusioni:

$$\alpha_1 = \text{"Non c'è pozzo in [1,2]”}$$

$$\alpha_2 = \text{"Non c'è pozzo in [2,2]”}.$$

I modelli corrispondenti alle formule α_1 e α_2 sono delimitati con linee tratteggiate rispettivamente nelle Figure 7.5(a) e 7.5(b).

Si può verificare quanto segue:

in ogni modello in cui KB è vera, lo è anche α_1 .

Quindi $KB \models \alpha_1$: non c'è pozzo in [1,2]. Possiamo anche verificare che:

in alcuni modelli in cui KB è vera, α_2 è falso.

³ Benché la figura mostri i modelli come mondi del wumpus parziali, in effetti essi non sono altro che assegnamenti dei valori *true* e *false* alle formule “c'è un pozzo in [1,2]” e così via. I modelli, in senso matematico, non hanno bisogno di contenere orribili mostri pelosi.

Quindi da KB non consegue logicamente α_2 : l'agente *non può* concludere che non ci sia un pozzo in [2,2] (e neppure che *vi sia*, peraltro).⁴

Questo esempio non solo illustra la conseguenza logica, ma mostra anche come la sua definizione può essere applicata per derivare conclusioni, ovvero per eseguire **inferenze logiche**. L'algoritmo di inferenza riportato nella Figura 7.5 è chiamato **model checking**, perché enumera tutti i possibili modelli per verificare che α sia vero in tutti quelli in cui è vera la KB , ovvero che $M(KB) \subseteq M(\alpha)$.

Per comprendere meglio conseguenza logica e inferenza, potreste provare a pensare che l'insieme di tutte le conseguenze della KB sia come un pagliaio e α sia un ago. La conseguenza logica è come dire che l'ago si trova nel pagliaio; l'inferenza equivale a trovarlo. Questa distinzione viene rappresentata nella notazione formale: se un algoritmo di inferenza i può derivare α da KB , scriviamo:

$$KB \vdash_i \alpha,$$

e si dice che “ α è derivato da KB attraverso i ” o anche “ i deriva α da KB ”.

Un algoritmo (o procedura) di inferenza che deriva solo formule che sono conseguenze logiche si dice **corretto**; si dice anche che **preserva la verità**. La correttezza è una proprietà decisamente auspicabile. Una procedura di inferenza non corretta, essenzialmente, genera formule a suo piacimento: è come dire che annuncia la scoperta di aghi inesistenti. È facile verificare che il model checking, quando è applicabile,⁵ è una procedura corretta.

Un'altra proprietà desiderabile è la **completezza**: un algoritmo di inferenza è completo se può derivare ogni formula che è conseguenza logica. Nei pagliai reali, che hanno un'estensione finita, è abbastanza ovvio che un esame sistematico può sempre decidere se l'ago vi è contenuto oppure no. In molte basi di conoscenza, però, il pagliaio delle conseguenze è infinito, e la completezza diventa una questione importante.⁶ Fortunatamente, esistono procedure di inferenza complete per logiche abbastanza espansive da gestire molte basi di conoscenza.

Abbiamo descritto un processo di ragionamento le cui conclusioni sono garantite vere in qualsiasi mondo in cui sono vere le premesse; in particolare, *se KB è vera nel mondo reale, allora ogni formula α derivata da KB con un procedimento di inferenza corretto è anch'essa vera nel mondo reale*. Così mentre un processo di inferenza lavora a livello di “sintassi”, ovvero di configurazioni fisiche interne come i bit nei registri o i pattern di segnali elettrici nel cervello, il processo *corrisponde* alla relazione esistente nel mondo reale laddove qualche aspetto accade⁷ in virtù del fatto che accade qualche altro aspetto del mondo reale. Questa corrispondenza tra mondo e rappresentazione è illustrata nella Figura 7.6.

L'ultimo aspetto da considerare è quello del **grounding** (che potremmo tradurre *radicamento* o *ancoramento*): il legame tra i processi di ragionamento logico e l'ambiente reale in cui si trova l'agente. In particolare, *come facciamo a sapere che la KB è vera nel mondo reale?* Dopo tutto, si tratta solo di “sintassi” nella testa dell'agente. Questa è una questione filosofica su cui sono stati scritti molti libri: la riprenderemo nel Capitolo 27. Una risposta semplice è che il legame è creato dai sensori dell'agente. Per esempio, il nostro agente nel mondo del wumpus ha un rilevatore di odori. Il programma agente crea una formula adeguata ogni volta che viene percepito un odore. Di conseguenza, ogni volta che quella formula è nella

**inferenza logica
model checking**

**correttezza
preservazione
della verità**

completezza

grounding

⁴ L'agente può però calcolare la *probabilità* che ci sia un pozzo in [2,2]; vedremo come nel Capitolo 12.

⁵ Il model checking funziona se lo spazio dei modelli è finito: ad esempio, nei mondi del wumpus di estensione limitata. Nell'aritmetica, d'altra parte, lo spazio dei modelli è infinito: anche se ci limitiamo a considerare numeri interi, ci sono sempre infinite coppie di valori per x e y tali che $x + y = 4$.

⁶ Confrontate questo caso con quello degli spazi di ricerca infiniti che abbiamo visto nel Capitolo 3, in cui la ricerca in profondità non è completa.

⁷ Come scrisse Wittgenstein (1922) nel suo famoso *Tractatus*: “Il mondo è tutto ciò che accade”.

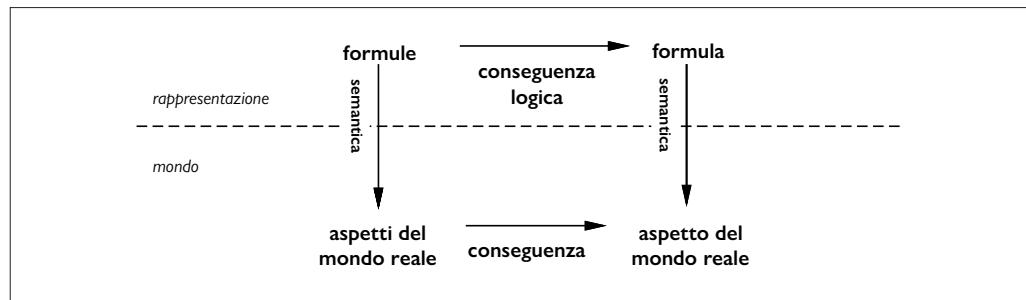


Figura 7.6 Le formule sono configurazioni fisiche dell'agente, e il ragionamento è il processo di costruzione di nuove configurazioni partendo dalle vecchie. Il ragionamento logico dovrebbe assicurare che le nuove configurazioni rappresentino aspetti del mondo che sono effettive conseguenze degli aspetti del mondo rappresentati dalle vecchie configurazioni.

base di conoscenza, è vera nel mondo reale. Il significato e la verità delle formule percettive sono così definite dai processi di percezione e di costruzione sintattica che le producono. E per quanto riguarda il resto della conoscenza dell'agente, per esempio circa il fatto che il wumpus causa la presenza di un pungente fetore nelle stanze adiacenti? Questa non è una rappresentazione diretta di una singola percezione ma una regola generale, forse derivata dall'esperienza percettiva ma nient'affatto identica a una formula che rappresenta tale esperienza. Regole generali come questa sono prodotte da un processo di costruzione di formule chiamato **apprendimento**, che sarà trattato nella Parte V. L'apprendimento è soggetto a errori: potrebbe darsi che i wumpus puzzino sempre *tranne il 29 febbraio degli anni bisestili*, giorno in cui fanno il bagno. Di conseguenza la *KB* potrebbe non essere vera nel mondo reale, ma questo non toglie che, se si adottano buone procedure di apprendimento, ci siano buone ragioni di essere ottimisti.

7.4 Logica proposizionale: una logica molto semplice

Introduciamo ora la **logica proposizionale** (o calcolo proposizionale), descrivendone la sintassi (la struttura delle formule) e la semantica (il modo in cui si determina il valore di verità delle formule). Su queste basi descriveremo un semplice algoritmo sintattico per l'inferenza logica che implementa la nozione semantica di **conseguenza logica**. Tutto ciò, naturalmente, nel mondo del wumpus.

7.4.1 Sintassi

formula atomica
simbolo proposizionale

formula complessa
connettivo logico

La **sintassi** della logica proposizionale definisce le formule accettabili. Le **formule atomiche** consistono di un singolo **simbolo proposizionale**. Ogni simbolo rappresenta una proposizione che può essere vera o falsa. Utilizziamo simboli che iniziano con una lettera maiuscola e possono contenere altre lettere o pedici, per esempio: *P*, *Q*, *R*, *W_{1,3}* e *RivoltoEst*. I nomi sono arbitrari, ma spesso li sceglieremo in modo da essere significativi per il lettore: per esempio, potremo scrivere *W_{1,3}* per indicare la proposizione che il wumpus si trova in [1,3] (ricordate sempre che i simboli sono *atomici*, questo significa che *W_{1,3}* non si può scomporre nei suoi componenti *W*, 1 e 3). Due simboli proposizionali hanno un valore prefissato: *True* è la proposizione sempre vera, *False* quella sempre falsa. Si possono costruire **formule complesse** (o composte) partendo da formule più semplici grazie all'uso delle parentesi e di operatori chiamati **connettivi logici**. Ci sono cinque connettivi di uso comune.

- ¬ (not): una formula come $\neg W_{1,3}$ è chiamata la **negazione** di $W_{1,3}$. Si può usare il termine **letterale (literal)** per indicare una formula atomica (**letterale positivo**) o una formula atomica negata (**letterale negativo**). negazione
letterale
- ∧ (and): una formula il cui connettivo principale è \wedge , come $W_{1,3} \wedge P_{3,1}$, prende il nome di **congiunzione**; le sue due parti si chiamano **congiunti**. Il simbolo \wedge ricorda la “A” di “And”. congiunzione
- ∨ (or): una formula in cui il connettivo principale è \vee , come $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, è una **disgiunzione**: le sue parti sono dette **disgiunti** – in questo esempio, i disgiunti sono $(W_{1,3} \wedge P_{3,1})$ e $W_{2,2}$. Storicamente, il simbolo \vee deriva dal latino “vel”, che significa “oppure”; per alcuni sarà più facile ricordarlo semplicemente come un \wedge rovesciato. disgiunzione
- \Rightarrow (implicazione): una formula come $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ si chiama **implicazione** (o condizionale). La sua **premessa o antecedente** è $(W_{1,3} \wedge P_{3,1})$ e la sua **conclusione o conseguente** è $\neg W_{2,2}$. Le implicazioni sono anche dette **regole** o asserzioni **if–then**. In altri testi per indicare l’implicazione si usa il simbolo insiemistico \supset o la freccia sottile \rightarrow . implicazione
premessa
conclusione
regola
- \Leftrightarrow (se e solo se): la formula $W_{1,3} \Leftrightarrow \neg W_{2,2}$ è una **equivalenza**, chiamata anche **bicondizionale**. bicondizionale

La Figura 7.7 fornisce una grammatica formale della logica proposizionale (la notazione BNF è spiegata nell’Appendice B). La grammatica BNF è arricchita con un elenco di precedenze degli operatori per eliminare ogni ambiguità quando si usano operatori multipli. L’operatore “not” (\neg) ha la precedenza più alta, quindi nella formula $\neg A \wedge B$ il connettivo \neg fornisce il legame più stretto, per cui la formula equivale a $(\neg A) \wedge B$ e non a $\neg(A \wedge B)$. In aritmetica vale lo stesso: $-2 + 4$ è 2, non -6. Quando opportuno, utilizziamo anche parentesi tonde e quadre per chiarire meglio la struttura della formula e migliorarne la leggibilità.

7.4.2 Semantica

Dopo aver specificato la sintassi della logica proposizionale, passiamo ora alla semantica. Con questo termine si intendono le regole usate per determinare il valore di verità di una formula nei confronti di un particolare modello. Nella logica proposizionale, un modello stabilisce semplicemente il **valore di verità** – *true* o *false* – di ogni simbolo proposizionale. Per esempio, se le formule nella base di conoscenza fanno uso dei simboli $P_{1,2}$, $P_{2,2}$, e $P_{3,1}$, un modello possibile è:

$$m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}.$$

Con tre simboli ci sono $2^3 = 8$ possibili modelli, esattamente quelli mostrati nella Figura 7.5. Notate, comunque, che i modelli sono oggetti puramente matematici senza alcuna relazione particolare con i mondi e i wumpus. $P_{1,2}$ è semplicemente un simbolo: potrebbe significare “c’è un pozzo nella posizione [1,2]” o “sarò a Parigi oggi e domani”.

valore di verità

<i>formula</i>	\rightarrow	<i>formulaAtomica</i> <i>formulaComplessa</i>
<i>formulaAtomica</i>	\rightarrow	<i>True</i> <i>False</i> <i>P</i> <i>Q</i> <i>R</i> ...
<i>formulaComplessa</i>	\rightarrow	(formula) $[\text{formula}]$
		$\neg \text{formula}$
		$(\text{formula} \wedge \text{formula})$
		$(\text{formula} \vee \text{formula})$
		$(\text{formula} \Rightarrow \text{formula})$
		$(\text{formula} \Leftrightarrow \text{formula})$
PRECEDENZA OPERATORI	:	$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Figura 7.7
Una grammatica delle formule della logica proposizionale in forma BNF (Backus–Naur Form), con le precedenze degli operatori dalla più alta alla più bassa.

La semantica della logica proposizionale deve specificare come si fa, dato un modello, a calcolare il valore di verità di *qualsiasi* formula. Questo viene fatto ricorsivamente. Tutte le formule sono costruite partendo da formule atomiche e applicando i cinque connettivi; di conseguenza dobbiamo specificare come calcolare la verità delle formule atomiche e di quelle costruite con ognuno dei connettivi. Per le formule atomiche è facile:

- *True* è vero in ogni modello e *False* è falso in ogni modello;
- il valore di verità di ogni altro simbolo proposizionale dev'essere specificato direttamente nel modello. Per esempio, nel modello m_1 che abbiamo definito precedentemente, $P_{1,2}$ è falso.

Per le formule complesseabbiamo cinque regole che valgono per qualsiasi sottoformula P e Q (atomiche o complesse) in ogni modello m (nel seguito “sse” significa “se e solo se”):

- $\neg P$ è vero sse P è falso in m .
- $P \wedge Q$ è vero sse P e Q sono entrambi veri in m .
- $P \vee Q$ è vero sse P o Q è vero in m .
- $P \Rightarrow Q$ è vero a meno che P sia vero e Q falso in m .
- $P \Leftrightarrow Q$ è vero sse P e Q sono entrambi veri o entrambi falsi in m .

tavola di verità

Queste regole possono anche essere espresse con **tavole di verità** che specificano i valori di verità di una formula complessa per ogni possibile configurazione dei suoi componenti. Le tavole di verità dei cinque connettivi logici sono fornite nella Figura 7.8. Usando queste tavole, il valore di verità di ogni formula s può essere calcolato per ogni modello m con un semplice processo di valutazione ricorsiva. Per esempio, la formula $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$, valutata in m_1 , corrisponde a $true \wedge (false \vee true) = true \wedge true = true$. Nell'Esercizio 7.TRUVE vi sarà richiesto di scrivere l'algoritmo PL-VERO?(s, m), che calcola il valore di verità di una formula della logica proposizionale s in un modello m .

Le tavole di verità per “and”, “or” e “not” corrispondono al significato intuitivo delle corrispondenti parole inglesi “e”, “oppure” e “non”. La principale causa di confusione sta nel fatto che la formula $P \vee Q$ è vera quando P è vera, o lo è Q , o anche quando lo sono *entrambe*. C'è un diverso connettivo chiamato “or esclusivo” (o “xor” in breve) che restituisce falso quando entrambi i disgiunti sono veri.⁸ Non c'è consenso generale sul simbolo da utilizzare per indicare l'or esclusivo; tra le scelte possibili vi sono $\dot{\vee}$, \neq o \oplus .

La tavola di verità per \Rightarrow potrebbe sembrare strana, perché non corrisponde al significato intuitivo di “ P implica Q ” o “se P , allora Q ”. Prima di tutto, occorre dire che la logica pro-

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Figura 7.8 Tavole di verità dei cinque connettivi logici. Volendo usare la tavola nel calcolare, per esempio, il valore di $P \vee Q$ quando P è vero e Q falso, per prima cosa cercate nelle due colonne a sinistra la riga in cui P vale *true* e Q *false* (in questo caso, la riga è la terza). Una volta trovata la riga giusta incrociate con la colonna $P \vee Q$ per vedere il risultato: *true*.

⁸ In latino si usano due parole distinte: *vel* indica l'or inclusivo e *aut* l'or esclusivo.

posizionale non richiede alcuna relazione di *causalità*, e a dire il vero nessuna relazione in assoluto, tra P e Q . L'enunciato “5 è dispari implica che Tokyo è la capitale del Giappone”, per quanto sia strano da pronunciare, è una formula vera della logica proposizionale. Un altro punto che può generare confusione è che ogni implicazione è vera se il suo antecedente è falso. Per esempio, “5 è pari implica che Sam è intelligente” è una formula vera, indipendentemente dall’effettivo quoziante intellettuale di Sam. Questa può sembrare una scelta bizzarra, ma si deve pensare che la formula “ $P \Rightarrow Q$ ” in effetti significa “se P è vero, allora sostengo che lo è anche Q . In caso contrario non faccio alcuna asserzione”. L’unico modo perché questa formula risulti falsa è che P sia vera ma Q falsa.

Il bicondizionale, $P \Leftrightarrow Q$, è vero quando lo sono sia $P \Rightarrow Q$ che $Q \Rightarrow P$. Spesso in questo caso si usa l’espressione “ P se e solo se Q ”. Molte regole del mondo del wumpus si esprimono bene usando \Leftrightarrow . Per esempio, affinché in una stanza ci sia movimento d’aria è *necessario* che in una delle stanze adiacenti ci sia un pozzo, ma d’altronde la brezza si può rilevare *solamente* in tal caso. Di conseguenza possiamo scrivere una formula bicondizionale come:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

dove $B_{1,1}$ significa che si percepisce una brezza in [1,1].

7.4.3 Una semplice base di conoscenza

Ora che abbiamo definito la semantica della logica proposizionale, possiamo costruire una base di conoscenza per il mondo del wumpus. Ci concentreremo prima sugli aspetti *immutabili* di tale mondo, lasciando quelli mutevoli a un successivo paragrafo. Per ora, ci servono i seguenti simboli per ogni posizione $[x, y]$:

$P_{x,y}$ è vero se esiste un pozzo in $[x, y]$.

$W_{x,y}$ è vero se esiste un wumpus in $[x, y]$, morto o vivo.

$B_{x,y}$ è vero se esiste una brezza in $[x, y]$.

$S_{x,y}$ è vero se esiste un fetore in $[x, y]$.

$L_{x,y}$ è vero se l’agente si trova nella locazione $[x, y]$.

Le formule che scrivremo saranno sufficienti per ricavare $\neg P_{1,2}$ (non ci sono pozzi in [1,2]), come abbiamo fatto in modo informale nel Paragrafo 7.3. Etichettiamo ogni formula con R_i in modo da potervi fare riferimento:

- In [1,1] non ci sono pozzi:

$$R_1 : \neg P_{1,1}.$$

- In una stanza si percepisce brezza se e solo se c’è un pozzo in una stanza adiacente. Questo dev’essere specificato per ogni locazione; per ora indichiamo solo quelle rilevanti:

$$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}).$$

- Le formule qui sopra sono vere in tutti i mondi del wumpus. Ora aggiungiamo le percezioni relative alla brezza raccolte nelle prime due stanze visitate dall’agente nello specifico mondo in cui si trova, ponendoci così nella situazione rappresentata nella Figura 7.3(b):

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}.$$

7.4.4 Una semplice procedura di inferenza

Il nostro scopo ora è decidere se, data una formula α , $KB \models \alpha$. Per esempio, $\neg P_{1,2}$ è conseguenza logica della nostra KB? Il nostro primo algoritmo per l’inferenza sarà un approccio

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	false	true	false	false	false
:	:	:	:	:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	<u>true</u>
false	true	false	false	false	false	true	true	true	true	true	true	<u>true</u>
false	true	false	false	true	false	false	true	false	false	true	true	false
:	:	:	:	:	:	:	:	:	:	:	:	:
true	false	true	true	false	true	false						

Figura 7.9 Una tavola di verità per la base di conoscenza fornita nel testo. KB è vera se sono vere tutte le formule da R_1 a R_5 , il che si verifica solo in 3 delle 128 righe (quelle sottolineate nella colonna più a destra). In tutte e tre $P_{1,2}$ è falsa, per cui si può dire che non c'è alcun pozzo in [1,2]. D'altra parte, in [2,2] potrebbe esserci un pozzo o no.

basato sul model checking che è un'implementazione diretta della definizione di conseguenza logica: dovremo enumerare esplicitamente i modelli e verificare che α sia vera in ogni modello in cui la KB lo è. I modelli sono rappresentati da un assegnamento dei valori *true* o *false* a ogni simbolo proposizionale. Ritornando al mondo del wumpus, i simboli sono $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$ e $P_{3,1}$. Con sette simboli, ci sono $2^7 = 128$ possibili modelli; in tre di essi KB è vera (Figura 7.9). In quei tre modelli $\neg P_{1,2}$ è vera, per cui possiamo star certi che non c'è pozzo in [1,2]. D'altra parte, $P_{2,2}$ è vera in due dei tre modelli e falsa in uno, quindi non possiamo ancora dire se in [2,2] c'è un pozzo oppure no.

La Figura 7.9 riproduce in una forma più precisa il ragionamento illustrato nella Figura 7.5. La Figura 7.10 mostra un algoritmo generale per calcolare le conseguenze logiche nella logica proposizionale. Come la RICERCA-BACKTRACKING citata nella Figura 6.5 del Capitolo 6, TV-CONSEGUE? esegue l'enumerazione ricorsiva di uno spazio finito di assegnamenti a simboli. L'algoritmo è **corretto**, dato che implementa direttamente la definizione di conseguenza logica, ed è **completo**, perché funziona con ogni base di conoscenza KB e formula α e termina sempre l'esecuzione: infatti il numero dei modelli da esaminare è sempre finito.

Naturalmente, dire che un numero è finito non significa che sia piccolo: se il numero totale di simboli contenuti in KB e α è n , i modelli saranno 2^n . La complessità temporale dell'algoritmo è quindi $O(2^n)$ (quella spaziale è solamente $O(n)$, perché l'enumerazione viene svolta in profondità). Più avanti vedremo che in molti casi esistono algoritmi molto più efficienti: sfortunatamente, la conseguenza logica proposizionale è co-NP-completa (ovvero, probabilmente non più semplice che NP-completa, cfr. Appendice A), perciò *ogni algoritmo di inferenza conosciuto per la logica proposizionale ha una complessità, nel caso pessimo, esponenziale nelle dimensioni dell'input*.

7.5 Dimostrazione di teoremi nella logica proposizionale

Fin qui abbiamo mostrato come determinare la conseguenza logica mediante il *model checking*: enumerando modelli e mostrando che la formula deve valere in tutti. In questo paragrafo mostriamo come la conseguenza logica può essere ottenuta tramite la **dimostrazione di teoremi**, applicando regole di inferenza direttamente alle formule della nostra base di conoscenza per costruire una dimostrazione della formula desiderata senza consultare

```

function TV-CONSEGUE?(KB,  $\alpha$ ) returns true oppure false
  inputs: KB, la base di conoscenza, una formula della logica proposizionale
             $\alpha$ , la query, una formula della logica proposizionale

  simboli  $\leftarrow$  una lista dei simboli proposizionali contenuti in KB e  $\alpha$ 
  return TV-VERIFICA-TUTTO(KB,  $\alpha$ , simboli, {})

function TV-VERIFICA-TUTTO(KB,  $\alpha$ , simboli, modello) returns true oppure false
  if VUOTO?(simboli) then
    if PL-VERO?(KB, modello) then return PLP-VERO?( $\alpha$ , modello)
    else return true // quando KB è false, restituisce sempre true
  else do
    P  $\leftarrow$  PRIMO(simboli); resto  $\leftarrow$  RESTO(simboli)
    return TV-VERIFICA-TUTTO(KB,  $\alpha$ , resto, modello  $\cup$  {P = true})
    and
    TV-VERIFICA-TUTTO(KB,  $\alpha$ , resto, modello  $\cup$  {P = false})

```

Figura 7.10 Un algoritmo che enumera una tavola di verità per determinare la conseguenza logica nella logica proposizionale (“TV” sta appunto per “tavola di verità”). PL-VERO? restituisce *true* se la formula è vera nel modello. La variabile *modello* rappresenta un modello parziale, in cui sono stati assegnati valori solo ad alcune variabili. La parola chiave **and** qui è un simbolo di funzione infissa nel linguaggio di programmazione in pseudocodice, non un operatore della logica proposizionale; richiede due argomenti e restituisce *true* o *false*.

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutatività di \wedge
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutatività di \vee
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associatività di \wedge
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associatività di \vee
$\neg(\neg\alpha) \equiv \alpha$	eliminazione della doppia negazione
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contrapposizione
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	eliminazione dell’implicazione
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	eliminazione del bicondizionale
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributività di \wedge su \vee
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributività di \vee su \wedge

Figura 7.11

Alcune equivalenze logiche standard. I simboli α , β e γ rappresentano formule arbitrarie della logica proposizionale.

modelli. Se il numero di modelli è elevato, ma la lunghezza della dimostrazione è ridotta, la dimostrazione di teoremi può essere più efficiente del model checking.

Prima di addentrarci nei dettagli degli algoritmi di dimostrazione di teoremi, ci serve qualche altro concetto relativo alla conseguenza logica. Il primo concetto è quello di **equivalenza logica**: due formule α e β sono logicamente equivalenti se sono vere nello stesso insieme di modelli. Questo si scrive $\alpha \equiv \beta$ (notate che \equiv è usato per fare asserzioni su formule, mentre \Leftrightarrow fa parte di una formula). Per esempio, possiamo mostrare facilmente con le tavole di verità che $P \wedge Q$ e $Q \wedge P$ sono logicamente equivalenti; altre equivalenze sono riportate nella Figura 7.11. Il ruolo di queste equivalenze è analogo a quello delle identità aritmetiche in

equivalenza logica

matematica. Una definizione alternativa dell’equivalenza è la seguente: due formule α e β sono equivalenti se e soltanto se ognuna di esse è conseguenza logica dell’altra:

$$\alpha \equiv \beta \quad \text{se e solo se} \quad \alpha \models \beta \text{ e } \beta \models \alpha.$$

**validità
tautologia**

Il secondo concetto è quello di **validità**. Una formula è valida se è vera in *tutti* i modelli. Per esempio, la formula $P \vee \neg P$ è valida. Le formule valide sono note anche come **tautologie** e sono *necessariamente* vere. Dato che la formula *True* è vera in tutti i modelli, ogni formula valida è logicamente equivalente a *True*. A che cosa servono le formule valide? Dalla nostra definizione di conseguenza logica possiamo derivare il **teorema di deduzione**, che era già noto agli antichi greci:

$$\text{date due formule qualsiasi } \alpha \text{ e } \beta, \alpha \models \beta \text{ se e solo se la formula } (\alpha \Rightarrow \beta) \text{ è valida.}$$

(L’Esercizio 7.5DEDU vi chiederà di dimostrarlo). Di conseguenza, possiamo decidere se $\alpha \models \beta$ controllando che $(\alpha \Rightarrow \beta)$ sia vera in ogni modello, che in sostanza è ciò che fa l’algoritmo di inferenza della Figura 7.10, oppure dimostrando che $(\alpha \Rightarrow \beta)$ è equivalente a *True*. Viceversa, il teorema di deduzione afferma che ogni formula di implicazione valida descrive un’inferenza legittima.

soddisfacibilità

L’ultimo concetto è la **soddisfacibilità**. Una formula è soddisfacibile se è vera in, o soddisfatta da, *qualche* modello. Per esempio, la base di conoscenza che abbiamo scritto in precedenza, $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$ è soddisfacibile perché, come si vede nella Figura 7.9, è vera in tre modelli. La soddisfacibilità può essere verificata enumerando i possibili modelli finché non se ne trova uno che soddisfa la formula. Il problema di determinare la soddisfacibilità delle formule della logica proposizionale, o problema **SAT**, è stato il primo problema di cui è stata dimostrata la NP-completezza. Nell’informatica molti problemi sono in realtà problemi di soddisfacibilità. Tutti i problemi di soddisfacimento di vincoli che abbiamo trattato nel Capitolo 6, per esempio, richiedono in sostanza di verificare se i vincoli sono soddisfacibili da qualche assegnamento delle variabili.

SAT

Naturalmente, validità e soddisfacibilità sono strettamente connesse: α è valida se e solo se $\neg\alpha$ è insoddisfacibile; di contro, α è soddisfacibile se e solo se $\neg\alpha$ non è valida. Un altro risultato utile è il seguente:

$$\alpha \models \beta \text{ se e solo se la formula } (\alpha \wedge \neg\beta) \text{ è insoddisfacibile.}$$

**dimostrazione
per assurdo
refutazione
contraddizione**

Dimostrare β partendo da α verificando l’insoddisfacibilità di $(\alpha \wedge \neg\beta)$ corrisponde esattamente alla tecnica matematica standard chiamata *dimostrazione per assurdo*, detta anche *dimostrazione per refutazione* o per **contraddizione**. Si assume che la formula β sia falsa e si dimostra che questo porta a una contraddizione con gli assiomi α (che sono noti). Questa contraddizione è esattamente ciò che si intende quando si dice che la formula $(\alpha \wedge \neg\beta)$ è insoddisfacibile.

**regola di inferenza
dimostrazione
Modus Ponens**

7.5.1 Inferenza e dimostrazioni

Questo paragrafo tratta **regole di inferenza** che possono essere applicate per derivare una **dimostrazione** o prova, ovvero una catena di conclusioni che portano all’obiettivo desiderato. La regola più nota è il **Modus Ponens** (dal latino, significa “modo che afferma”), che si scrive come segue:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}.$$

La notazione significa che, ogni volta che sono date le formule $\alpha \Rightarrow \beta$ e α , allora si può inferire la formula β . Per esempio, se sono date le formule $(WumpusDavanti \wedge WumpusVivo) \Rightarrow ScagliaFreccia$ e vale $(WumpusDavanti \wedge WumpusVivo)$, allora si può inferire *ScagliaFreccia*.

Un'altra utile regola di inferenza è l'**eliminazione degli and** in base alla quale, data una congiunzione, si può inferire uno qualsiasi dei congiunti:

$$\frac{\alpha \wedge \beta}{\alpha}.$$

Per esempio, da $(WumpusDavanti \wedge WumpusVivo)$ si può inferire $WumpusVivo$.

Considerando i possibili valori di verità di α e β , si può facilmente mostrare una volta per tutte che Modus Ponens ed eliminazione degli and sono corrette. Queste regole possono quindi essere utilizzate in tutte le istanze a cui si applicano, producendo inferenze corrette senza bisogno di enumerare i modelli.

Tutte le equivalenze logiche della Figura 7.11 possono essere usate come regole di inferenza. Per esempio, l'equivalenza per l'eliminazione del bicondizionale dà origine alle due regole di inferenza:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Leftrightarrow \beta) \wedge (\beta \Leftrightarrow \alpha)} \text{ e } \frac{(\alpha \Leftrightarrow \beta) \wedge (\beta \Leftrightarrow \alpha)}{\alpha \Leftrightarrow \beta}.$$

Non tutte le regole di inferenza funzionano, come questa, in entrambe le direzioni: per esempio, non è possibile eseguire Modus Ponens all'indietro per dedurre $\alpha \Rightarrow \beta$ e α da β .

Vediamo ora come si possono utilizzare queste regole di inferenza ed equivalenze nel mondo del wumpus. La base di conoscenza iniziale contiene le formule da R_1 a R_5 , e vogliamo dimostrare $\neg P_{1,2}$, ovvero che non c'è alcun pozzo in [1,2].

1. Applichiamo l'eliminazione del bicondizionale a R_2 per ottenere:

$$R_6: (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}).$$

2. Applichiamo l'eliminazione degli and a R_6 per ottenere:

$$R_7: ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}).$$

3. L'equivalenza logica della contrapposizione fornisce:

$$R_8: (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})).$$

4. Applichiamo Modus Ponens con R_8 e la percezione R_4 (cioè $\neg B_{1,1}$), ottenendo:

$$R_9: \neg(P_{1,2} \vee P_{2,1}).$$

5. Applichiamo la regola di De Morgan, arrivando alla conclusione:

$$R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}.$$

Ovvero, non ci sono pozzi né in [1,2] né in [2,1].

Potremmo applicare uno qualsiasi degli algoritmi di ricerca del Capitolo 3 per trovare una sequenza di passi che costituisca una dimostrazione come questa. Dobbiamo soltanto definire un problema di dimostrazione nel modo seguente:

- **STATO INIZIALE:** la base di conoscenza iniziale.
- **AZIONI:** l'insieme delle azioni consiste di tutte le regole di inferenza applicate a tutte le formule che corrispondono (“fanno match”) alla metà superiore della regola di inferenza.
- **RISULTATO:** il risultato di un'azione è l'aggiunta della formula nella metà inferiore della regola di inferenza.
- **OBIETTIVO:** l'obiettivo è uno stato che contiene la formula che stiamo cercando di dimostrare.

La ricerca di dimostrazioni, quindi, è un'alternativa all'enumerazione di modelli. In molti casi pratici, *trovare una dimostrazione può essere molto efficiente semplicemente perché si possono ignorare le proposizioni irrilevanti, indipendentemente dal loro numero*. Per esempio, la dimostrazione appena fornita per $\neg P_{1,2} \wedge \neg P_{2,1}$ non fa alcuna menzione delle proposizioni $B_{2,1}$, $P_{1,1}$, $P_{2,2}$ o $P_{3,1}$, che possono essere ignorate in tutta sicurezza. Infatti la pro-



posizione obiettivo $P_{1,2}$ compare solo nella formula R_2 , e le altre proposizioni in R_2 compiono solo in R_4 e R_2 ; ne consegue che R_1 , R_3 e R_5 non hanno alcun effetto sulla dimostrazione. Lo stesso ragionamento varrebbe anche se dovessimo aggiungere un milione di altre formule alla base di conoscenza: il semplice algoritmo della tavola di verità, al contrario, sarebbe presto sopraffatto dalla crescita esponenziale del numero di modelli.

monotonicità

Un’ultima proprietà dei sistemi logici è la **monotonicità**, che afferma che un insieme di formule che sono conseguenze logiche può solo *aumentare* man mano che si aggiunge informazione alla base di conoscenza.⁹ Per qualsiasi coppia di formule α e β ,

$$\text{se } KB \models \alpha \text{ allora } KB \wedge \beta \models \alpha.$$

Supponiamo per esempio che la base di conoscenza contenga l’asserzione aggiuntiva β che nel mondo esistono esattamente otto pozzi. Questa conoscenza potrebbe aiutare l’agente a trarre conclusioni *aggiuntive*, ma non potrà mai invalidare una conclusione α già dedotta, come il fatto che non c’è alcun pozzo nella posizione [1,2]. La monotonicità significa che le regole di inferenza possono essere applicate non appena si trovano nella base di conoscenza le premesse necessarie; le conclusioni di tali regole dovranno essere vere *indipendentemente dal resto delle formule contenute nella base di conoscenza*.

7.5.2 Dimostrazione per risoluzione

Abbiamo detto che le regole di inferenza viste fin qui sono *corrette*, ma non abbiamo discusso la questione della *completezza* degli algoritmi di inferenza che le utilizzano. Algoritmi come la ricerca ad approfondimento iterativo (cfr. Figura 3.12 nel Capitolo 3) sono completi nel senso che arriveranno a qualsiasi obiettivo raggiungibile, ma se le regole di inferenza disponibili sono inadeguate, l’obiettivo non sarà raggiungibile affatto: in altre parole, non esisterà alcuna dimostrazione che usi solo quelle regole. Per esempio, senza la regola di eliminazione del bicondizionale non sarebbe più possibile fare la dimostrazione riportata sopra. In questo paragrafo introdurremo una singola regola di inferenza, la **risoluzione**, che unita a qualsiasi algoritmo di ricerca completo dà luogo a un algoritmo di inferenza completo.

Cominceremo ad applicare una versione semplificata della regola di risoluzione nel mondo del wumpus. Consideriamo i passi che hanno portato alla Figura 7.4(a): l’agente ritorna da [2,1] a [1,1] e da lì passa in [1,2] dove percepisce una forte puzza, ma nessun movimento d’aria. Aggiungiamo alla base di conoscenza i seguenti fatti:

$$\begin{aligned} R_{11} : & \neg B_{1,2}. \\ R_{12} : & B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}). \end{aligned}$$

Applicando lo stesso processo che in precedenza ci ha portato a R_{10} , possiamo derivare l’assenza di pozzi in [2,2] e [1,3] (ricordate che sappiamo già che non c’è alcun pozzo in [1,1]):

$$\begin{aligned} R_{13} : & \neg P_{2,2}. \\ R_{14} : & \neg P_{1,3}. \end{aligned}$$

Possiamo anche applicare l’eliminazione del bicondizionale a R_3 , seguita dal Modus Ponens con R_5 , per dedurre il fatto che ci dev’essere un pozzo in [1,1], [2,2] o [3,1]:

$$R_{15} : P_{1,1} \vee P_{2,2} \vee P_{3,1}.$$

risolvente

Ora giungiamo alla prima applicazione della regola di risoluzione: la formula atomica $\neg P_{2,2}$ in R_{13} *risolve con* la formula atomica $P_{2,2}$ in R_{15} per fornire il **risolvente**:

$$R_{16} : P_{1,1} \vee P_{3,1}.$$

⁹ Le logiche **non monotone**, che violano la proprietà della monotonicità, esprimono una proprietà comune del ragionamento umano: la possibilità di cambiare idea. Le discuteremo nel Paragrafo 10.6.

In linguaggio naturale possiamo dire che se c'è un pozzo in [1,1], [2,2] o [3,1], e non si trova in [2,2], allora dev'essere in [1,1] o [3,1]. In modo analogo, la formula atomica $\neg P_{1,1}$ in R_1 risolve con $P_{1,1}$ in R_{16} per dare:

$$R_{17} : \quad P_{3,1}.$$

Ovvero: se c'è un pozzo in [1,1] o [3,1] e non si trova in [1,1], allora dev'essere in [3,1]. Questi ultimi due passi inferenziali sono esempi della regola di inferenza di **risoluzione unitaria**.

risoluzione unitaria

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k}$$

dove ogni ℓ è un letterale e ℓ_i e m sono **letterali complementari** (cioè, uno è la negazione dell'altro). La regola di risoluzione unitaria, quindi, prende una **clausola** – cioè una disunzione di letterali – e un letterale e produce una nuova clausola. Notate che un singolo letterale può essere considerato come una disunzione formata da un solo letterale, che in questo caso dà origine a una **clausola unitaria**.

La regola di risoluzione unitaria può essere generalizzata nella regola di **risoluzione**:

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

**letterali
complementari
clausola**

**clausola unitaria
risoluzione**

dove ℓ_i e m_j sono letterali complementari. In definitiva, la risoluzione prende due clausole e ne produce una nuova che contiene tutti i letterali delle due clausole originali *tranne* i due complementari. Per esempio, nel nostro caso:

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}.$$

Possiamo risolvere soltanto una coppia di letterali complementari per volta. Per esempio, possiamo risolvere P e $\neg P$ per dedurre:

$$\frac{P \vee \neg Q \vee R, \quad \neg P \vee Q}{\neg Q \vee Q \vee R},$$

C'è un altro aspetto tecnico che riguarda la regola di risoluzione: la clausola risultante dovrebbe contenere solo una copia di ogni letterale.¹⁰ La rimozione delle eventuali copie di letterali è chiamata **fattorizzazione**. Per esempio, se risolviamo $(A \vee B)$ con $(A \vee \neg B)$ otteniamo $(A \vee A)$, che viene ridotto ad A mediante fattorizzazione.

fattorizzazione

La *correttezza* della regola di risoluzione può essere facilmente dimostrata considerando il letterale ℓ_i che è il complementare del letterale m_j nell'altra clausola. Se ℓ_i è vero, allora m_j è falso, e quindi $m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n$ dev'essere vero, perché $m_1 \vee \dots \vee m_n$ è dato. Se ℓ_i è falso, allora $\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k$ dev'essere vero, perché $\ell_1 \vee \dots \vee \ell_k$ è dato. È chiaro che ℓ_i è vero oppure falso, per cui deve verificarsi l'una o l'altra di queste conclusioni, esattamente come afferma la regola di risoluzione.

L'aspetto più sorprendente della regola di risoluzione è che rappresenta la base di una famiglia di procedure di inferenza *complete*. Un dimostratore di teoremi basato sulla risoluzione può, per ogni coppia di formule α e β della logica proposizionale, decidere se $\alpha \models \beta$. I prossimi due sottoparagrafi mostreranno come.

¹⁰ Se una clausola viene considerata come un insieme di letterali, la restrizione è automaticamente rispettata. Usare la notazione degli insiemi per le clausole rende la regola di risoluzione molto più pulita, anche se richiede l'introduzione di notazione aggiuntiva.

<i>FormulaCNF</i>	\rightarrow	<i>Clausola</i> ₁ $\wedge \cdots \wedge$ <i>Clausola</i> _{<i>n</i>}
<i>Clausola</i>	\rightarrow	<i>Letterale</i> ₁ $\vee \cdots \vee$ <i>Letterale</i> _{<i>m</i>}
<i>Fatto</i>	\rightarrow	<i>Simbolo</i>
<i>Letterale</i>	\rightarrow	<i>Simbolo</i> \neg <i>Simbolo</i>
<i>Simbolo</i>	\rightarrow	<i>P</i> <i>Q</i> <i>R</i> ...
<i>FormaClausolaHorn</i>	\rightarrow	<i>FormaClausolaDefinita</i> <i>FormaClausolaObiettivo</i>
<i>FormaClausolaDefinita</i>	\rightarrow	(<i>Simbolo</i> ₁ $\wedge \cdots \wedge$ <i>Simbolo</i> _{<i>l</i>}) \Rightarrow <i>Simbolo</i>
<i>FormaClausolaObiettivo</i>	\rightarrow	(<i>Simbolo</i> ₁ $\wedge \cdots \wedge$ <i>Simbolo</i> _{<i>l</i>}) \Rightarrow <i>False</i>

Figura 7.12 Una grammatica per forma normale congiuntiva, clausole di Horn e clausole definite. Una clausola CNF tale che $\neg A \vee \neg B \vee C$ può essere scritta in forma di clausola definita come $A \wedge B \Rightarrow C$.

forma normale
congiuntiva
CNF

Forma normale congiuntiva

La regola di risoluzione si applica solo a clausole (disgiunzioni di letterali), ragion per cui sembrerebbe utilizzabile solo su basi di conoscenza e query composte da clausole. Come è possibile, allora, che possa portare a una procedura di inferenza completa per l'intera logica proposizionale? La risposta è che *ogni formula della logica proposizionale è logicamente equivalente a una congiunzione di clausole*. Una formula così espressa viene detta in **forma normale congiuntiva** o, con l'acronimo inglese, CNF (cfr. la Figura 7.12). Ora descriviamo una procedura di conversione in CNF; per illustrarla, convertiamo la formula R_2 : $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$. Ecco la sequenza dei passi:

1. eliminiamo \Leftrightarrow , rimpiazzando $\alpha \Leftrightarrow \beta$ con $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$:

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1});$$

2. eliminiamo \Rightarrow , rimpiazzando $\alpha \Rightarrow \beta$ con $\neg \alpha \vee \beta$:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1});$$

3. CNF richiede che \neg si applichi solo ai letterali, per cui lo “muoviamo all'interno” utilizzando ripetutamente le seguenti equivalenze dalla Figura 7.11:

$$\neg(\neg \alpha) \equiv \alpha \text{ (eliminazione della doppia negazione)}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \wedge \neg \beta) \text{ (De Morgan)}$$

$$\neg(\alpha \vee \beta) \equiv (\neg \alpha \vee \neg \beta) \text{ (De Morgan)}$$

In quest'esempio ci basta applicare una sola volta la terza regola:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1});$$

4. ora abbiamo una formula che contiene operatori nidificati \wedge e \vee applicati a letterali. Sfruttiamo la legge di distributività della Figura 7.11, distribuendo \vee su \wedge ovunque possibile:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}).$$

La formula originale è ora in CNF, come congiunzione di tre clausole. La sua lettura è molto più ardua, ma può essere usata come input per una procedura di risoluzione.

Un algoritmo di risoluzione

Le procedure di inferenza basate sulla risoluzione sfruttano il principio di dimostrazione per assurdo che abbiamo menzionato all'inizio del Paragrafo 7.5. Questo significa che, per dimostrare che $KB \models \alpha$, dimostriamo che $(KB \wedge \neg \alpha)$ è insoddisfacibile. Per far questo dobbiamo giungere a una contraddizione.

```

function PL-RISOLUZIONE( $KB, \alpha$ ) returns true oppure false
  inputs:  $KB$ , la base di conoscenza, una formula della logica proposizionale
             $\alpha$ , la query, una formula della logica proposizionale

   $clausole \leftarrow$  l'insieme di clausole nella rappresentazione CNF di  $KB \wedge \neg\alpha$ 
   $nuove \leftarrow \{ \}$ 
  loop do
    for each coppia di clausole  $C_i, C_j$  in  $clausole$  do
       $risolventi \leftarrow$  PL-RISOLVI( $C_i, C_j$ )
      if  $risolventi$  contiene la clausola vuota then return true
       $nuove \leftarrow nuove \cup risolventi$ 
    if  $nuove \subseteq clausole$  then return false
     $clausole \leftarrow clausole \cup nuove$ 

```

Figura 7.13 Un semplice algoritmo di risoluzione per la logica proposizionale. La funzione PL-RISOLVI restituisce l'insieme di tutte le possibili clausole ottenute risolvendo i due input.

Un algoritmo di risoluzione è mostrato nella Figura 7.13. Per prima cosa si converte $(KB \wedge \neg\alpha)$ in CNF; quindi si applica la regola di risoluzione alle clausole risultanti. Ogni coppia che contiene letterali complementari è risolta per produrre una nuova clausola che viene aggiunta all'insieme (se non vi è già presente). Il processo continua finché non si verifica una delle due seguenti possibilità:

- non è più possibile aggiungere alcuna clausola, nel qual caso α non è conseguenza logica di KB ;
- la risoluzione applicata a due clausole dà come risultato la clausola *vuota*, nel qual caso da KB consegue logicamente α .

La clausola vuota, una disgiunzione senza alcun disgiunto, è equivalente a *False* perché una disgiunzione è vera solo se è vero almeno uno dei disgiunti. Inoltre, la clausola vuota può avere origine solo dalla risoluzione di due clausole unitarie complementari come P e $\neg P$.

Possiamo applicare la procedura di risoluzione a un'inferenza molto semplice nel mondo del wumpus. Quando l'agente si trova in [1,1] non percepisce alcun spostamento d'aria, per cui non ci possono essere pozzi nelle stanze adiacenti. La base di conoscenza relativa è

$$KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

e vogliamo dimostrare α che corrisponde a, poniamo, $\neg P_{1,2}$. Convertendo $(KB \wedge \neg\alpha)$ in CNF, otteniamo le clausole riportate nella riga superiore della Figura 7.14. La riga inferiore della figura mostra le clausole ottenute risolvendo le coppie nella parte superiore. Quando risolviamo $P_{1,2}$ con $\neg P_{1,2}$ otteniamo la clausola vuota, indicata con un piccolo quadratino. L'analisi della Figura 7.14 rivela che molti passi di risoluzione non hanno alcuna utilità: per esempio, la clausola $B_{1,1} \vee \neg B_{1,1} \vee P_{1,2}$ è equivalente a *True* $\vee P_{1,2}$ che a sua volta è equivalente a *True*. Dedurre che *True* è vero non rappresenta un grande risultato; di conseguenza, ogni clausola in cui compaiono due letterali complementari può essere scartata.

Completezza della risoluzione

Per concludere la nostra discussione sulla risoluzione, dimostriamo ora che PL-RISOLUZIONE è completo. Per fare ciò introduciamo la **chiusura della risoluzione** $RC(S)$ di un insieme di clausole S , che è l'insieme di tutte le clausole derivabili dall'applicazione ripetuta della regola

chiusura della
risoluzione

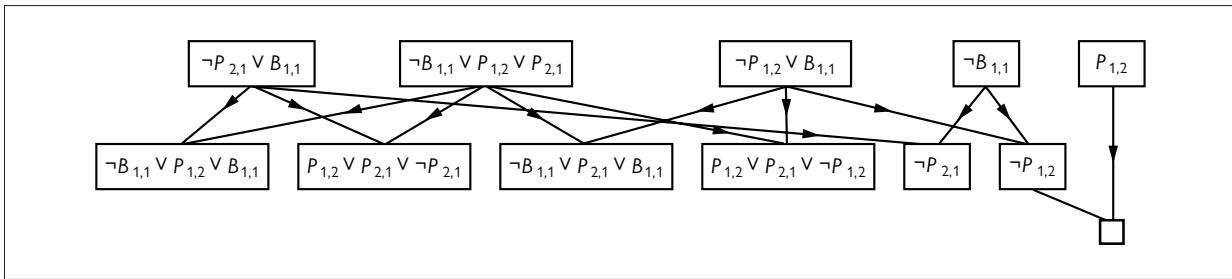


Figura 7.14 Applicazione parziale di PL-RISOLUZIONE a una semplice inferenza nel mondo del wumpus per dimostrare la query $\neg P_{1,2}$. Ognuna delle quattro clausole più a sinistra nella riga superiore è accoppiata con ognuna delle altre tre, e si applica la regola di risoluzione per ottenere le clausole nella riga inferiore. Vediamo che la terza e la quarta clausola nella riga superiore si combinano per dare come risultato la clausola $\neg P_{1,2}$, che viene poi risolta con $P_{1,2}$ ottenendo la clausola vuota, a indicare che la query è dimostrata.

di risoluzione alle clausole in S o a quelle da loro derivate. La chiusura della risoluzione è proprio quello che viene calcolato da **PL-RISOLUZIONE** come valore finale della variabile *clausole*. È facile vedere che $RC(S)$ dev'essere finita: grazie al passo di fattorizzazione, esiste solo un numero finito di clausole distinte che possono essere costruite con i simboli P_1, \dots, P_k di S . Ne consegue che **PL-RISOLUZIONE** termina sempre l'esecuzione.

Il teorema di completezza per la risoluzione nella logica proposizionale è chiamato **teorema di risoluzione ground**:

teorema di risoluzione ground

Se un insieme di clausole è insoddisfacibile, la sua chiusura della risoluzione contiene la clausola vuota.

Questo teorema si dimostra per contrapposizione: se la chiusura $RC(S)$ non contiene la clausola vuota, S è soddisfacibile. In effetti, possiamo costruire un modello di S assegnando adeguati valori di verità per P_1, \dots, P_k . La procedura di costruzione è la seguente.

Per i da 1 a k ,

- se in $RC(S)$ c'è una clausola che contiene il letterale $\neg P_i$ e tutti i suoi altri letterali sono falsi nell'assegnamento scelto per P_1, \dots, P_{i-1} , assegna *false* a P_i ;
 - altrimenti, assegna *true* a P_i .

Questo assegnamento di P_1, \dots, P_k è un modello di S . Per vederlo, ipotizziamo l'opposto, ovvero che, in qualche fase i della sequenza, l'assegnazione del simbolo P_i faccia diventare falsa una clausola C . Perché accada ciò, deve essersi verificato il fatto che *tutti* gli altri letterali in C sono già diventati falsi per assegnamenti di P_1, \dots, P_{i-1} . Quindi C ora deve apparire come $(\text{false} \vee \text{false} \vee \dots \vee \text{false} \vee P_i)$ o come $(\text{false} \vee \text{false} \vee \dots \vee \text{false} \vee \neg P_i)$. Se una sola di queste due è in $RC(S)$, l'algoritmo assegnerà il valore di verità appropriato a P_i per rendere vera C , perciò C può essere reso falso soltanto se *entrambe* queste clausole sono in $RC(S)$. Poiché $RC(S)$ è chiusa rispetto alla risoluzione, conterrà il risolvente di queste due clausole, e tale risolvente avrà tutti i letterali già falsi per gli assegnamenti di P_1, \dots, P_{i-1} . Questo contraddice la nostra ipotesi che la prima clausola falsa appaia nella fase i . Abbiamo quindi dimostrato che la costruzione non rende mai falsa una clausola in $RC(S)$, cioè che produce un modello di $RC(S)$. Infine, poiché S è contenuto in $RC(S)$, qualsiasi modello di $RC(S)$ è un modello di S stesso.

7.5.3 Clausole di Horn e clausole definite

La completezza della risoluzione la rende un metodo di inferenza molto importante. In molte applicazioni pratiche, tuttavia, non è necessario utilizzarne tutta la potenza. Alcune basi di conoscenza del mondo reale soddisfano certe restrizioni sulla forma delle formule

che contengono, e ciò consente di utilizzare un algoritmo di inferenza più ristretto ed efficiente.

Una forma ristretta è la **clausola definita**, una disgiunzione di letterali di cui *esattamente uno è positivo*. Per esempio, la clausola $(\neg L_{1,1} \vee \neg Brezza \vee B_{1,1})$ è una clausola definita, mentre $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ non lo è, poiché ha due letterali positivi.

Leggermente più generale è la **clausola di Horn**, una disgiunzione di letterali in cui *al massimo uno dei letterali è positivo*. Tutte le clausole definite sono clausole di Horn, come anche le clausole senza alcun letterale positivo, che si chiamano **clausole obiettivo**. Le clausole di Horn sono chiuse rispetto alla risoluzione: se si risolvono due clausole di Horn, si ottiene ancora una clausola di Horn. Esistono poi le formule k -CNF, che sono formule CNF in cui ogni clausola ha al più k letterali.

Le basi di conoscenza contenenti soltanto clausole definite sono interessanti per tre ragioni.

1. Ogni clausola definita può essere scritta come un'implicazione la cui premessa è una congiunzione di letterali positivi e la cui conclusione è un singolo letterale positivo (cfr. Esercizio 7.DISJ). Per esempio, la clausola definita $(\neg L_{1,1} \vee \neg Brezza \vee B_{1,1})$ può essere scritta come $(L_{1,1} \wedge Brezza) \Rightarrow B_{1,1}$. Nella forma di implicazione, la formula è più facile da leggere: dice che se l'agente si trova in [1,1] e percepisce uno spostamento d'aria, allora la stanza [1,1] è ventosa. Nella forma di Horn, la premessa si chiama **corpo** e la conclusione **testa**. Una formula costituita da un singolo letterale positivo, come $L_{1,1}$, viene chiamata **fatto**. Anch'essa può essere scritta in forma di implicazione come $True \Rightarrow L_{1,1}$, ma è più semplice scrivere soltanto $L_{1,1}$.
2. L'inferenza sulle clausole di Horn può essere svolta mediante gli algoritmi di **concatenazione in avanti** e **concatenazione all'indietro**, che vedremo tra poco. Entrambi gli algoritmi sono molto chiari per gli esseri umani, che trovano i passi di inferenza naturali e facili da seguire. Questo tipo di inferenza è la base della **programmazione logica**, discussa nel Capitolo 9.
3. Con le clausole di Horn è possibile determinare la conseguenza logica in un tempo che cresce *linearmente* con la dimensione della base di conoscenza: una piacevole sorpresa.

clausola definita

clausola di Horn

clausole obiettivo

corpo
testa
fatto

**concatenazione
in avanti**
**concatenazione
all'indietro**

7.5.4 Concatenazione in avanti e concatenazione all'indietro

L'algoritmo di concatenazione in avanti PL-CA-CONSEGUE?(KB, q) determina se un singolo simbolo proposizionale q – la query – è conseguenza logica di una base di conoscenza composta da clausole definite. L'algoritmo comincia dai fatti conosciuti (letterali positivi) nella base di conoscenza. Se tutte le premesse di un'implicazione sono verificate, la sua conclusione è aggiunta all'insieme dei fatti noti. Per esempio, se $L_{1,1}$ e $Brezza$ sono fatti conosciuti e nella base di conoscenza è presente la formula $(L \wedge Brezza) \Rightarrow B_{1,1}$, allora gli si può aggiungere $B_{1,1}$. Questo processo continua finché non viene aggiunta la stessa query q o non è più possibile effettuare alcuna inferenza. L'algoritmo è mostrato nella Figura 7.15; la sua caratteristica fondamentale è la complessità temporale lineare.

Il modo migliore di comprendere l'algoritmo è attraverso un esempio e una figura. La Figura 7.16(a) mostra una semplice base di conoscenza composta da clausole definite i cui fatti conosciuti sono A e B . La Figura 7.16(b) rappresenta la stessa base di conoscenza disegnata in forma di **grafo AND-OR** (cfr. Capitolo 4). Nei grafi AND-OR, più collegamenti uniti da un arco indicano una congiunzione (ogni collegamento dev'essere dimostrato), mentre collegamenti multipli senza arco indicano una disgiunzione (basta dimostrare uno qualsiasi dei collegamenti). È facile capire come funziona la concatenazione in avanti guardando il grafo. Partendo dalle foglie conosciute (qui, A e B) l'inferenza si propaga nel grafo il più lontano possibile. Ogni volta che si incontra una congiunzione, la propagazione attende finché non sono noti tutti i congiunti. Incoraggiamo i lettori a seguire passo passo l'esempio sul grafo.

```
function PL-CA-CONSEGUE?(KB, q) returns true oppure false
```

inputs: KB, la base di conoscenza, un insieme di clausole proposizionali definite

q, la query, un simbolo proposizionale

conto \leftarrow una tabella, dove $\text{conto}[c]$ è il numero di simboli della premessa della clausola c

inferiti \leftarrow una tabella, dove $\text{inferiti}[s]$ è inizialmente false per tutti i simboli

coda \leftarrow una coda di simboli, che contiene inizialmente quelli noti come veri nella KB

while coda non è vuota **do**

$p \leftarrow \text{POP}(\text{coda})$

if $p = q$ **then return** true

if $\text{inferiti}[p] = \text{false}$ **then**

$\text{inferiti}[p] \leftarrow \text{true}$

for each clausola c in KB dove p è in $c.\text{PREMESSA}$ **do**

 decrementa $\text{conto}[c]$

if $\text{conto}[c] = 0$ **then aggiungi** $c.\text{CONCLUSIONE}$ a coda

return false

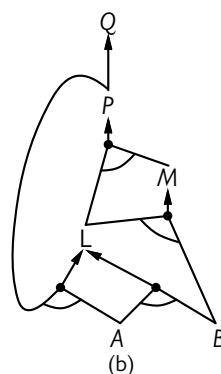
Figura 7.15 L’algoritmo di concatenazione in avanti per la logica proposizionale. La coda tiene traccia dei simboli che sono noti come veri ma non ancora “considerati.” La tabella *conto* tiene traccia di quante premesse di ogni implicazione non sono ancora dimostrate. Ogni volta che un nuovo simbolo p della coda è considerato, il conto viene ridotto di uno per ogni implicazione in cui appare la premessa p (facilmente individuata in tempo costante, con un’indicizzazione appropriata). Se un conto arriva a zero, significa che tutte le premesse di un’implicazione sono note, dimodoché la sua conclusione può essere aggiunta alla coda. Infine, dobbiamo tener traccia dei simboli considerati; un simbolo non dev’essere aggiunto nuovamente alla coda se è già compreso nell’insieme dei simboli inferiti. Questo permette di evitare di svolgere elaborazioni inutili e previene cicli infiniti nel caso esistano coppie di implicazioni come $P \Rightarrow Q$ e $Q \Rightarrow P$.

Figura 7.16

- (a) Un insieme di clausole definite.
- (b) Il corrispondente grafo AND-OR.

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ A \\ B \end{aligned}$$

(a)



(b)

È facile vedere che la concatenazione in avanti è **corretta**: ogni inferenza sostanzialmente è un’applicazione del Modus Ponens. L’algoritmo è anche **completo**: ogni formula atomica conseguenza logica sarà derivata. Il modo più facile di verificare quest’ultima affermazione è considerare lo stato finale della tabella *inferiti* dopo che l’algoritmo ha raggiunto un **punto fisso** oltre il quale non è più possibile inferire nulla. La tabella contiene il valore *true* per ogni simbolo inferito durante il processo e *false* per tutti gli altri simboli. Possiamo così con-



siderare la tabella alla stregua di un modello logico; inoltre, *ogni clausola definita presente nella KB originale è vera in questo modello*. Per verificare questo punto basta assumere l'ipotesi opposta, e cioè che qualche clausola $a_1 \wedge \dots \wedge a_k \Rightarrow b$ sia falsa nel modello. Allora la premessa $a_1 \wedge \dots \wedge a_k$ dev'essere vera, e b falsa: ma questo contraddice l'assunto che l'algoritmo abbia raggiunto un punto fisso, poiché ora potremmo aggiungere b alla KB. Possiamo quindi concludere che l'insieme di formule atomiche inferite in corrispondenza del punto fisso definisce un modello della KB originale. Ma ogni formula atomica q conseguenza logica della KB dev'essere vera in tutti i suoi modelli, e in particolare in questo. Ne consegue che l'algoritmo avrà necessariamente inferito ogni formula q che sia conseguenza logica.

La concatenazione in avanti è un esempio del concetto generale di **ragionamento guidato dai dati**, un tipo di ragionamento in cui l'attenzione parte dai fatti conosciuti. All'interno di un agente quest'approccio può essere usato per derivare conclusioni partendo dalle percezioni in input, spesso anche senza avere in mente un quesito specifico. Per esempio, l'agente del mondo del wumpus potrebbe riferire (TELL) le sue percezioni alla base di conoscenza usando un algoritmo di concatenazione in avanti incrementale in cui nuovi fatti possono essere aggiunti alla sua "coda" per dare inizio a nuove inferenze. Negli esseri umani, man mano che giunge nuova informazione, avviene una certa quantità di ragionamento guidato dai dati. Per esempio, se sono a casa e sento che ha cominciato a piovere, potrei pensare che probabilmente il picnic sarà cancellato. Tuttavia con ogni probabilità non mi verrà in mente che il diciassettesimo petalo della rosa più grande nel giardino del mio vicino si bagnerà; gli esseri umani infatti tengono la concatenazione in avanti sotto stretto controllo, per non essere sopraffatti da un numero enorme di conseguenze irrilevanti.

**ragionamento
guidato dai dati**

L'algoritmo di concatenazione all'indietro, come suggerisce il nome, parte dalla query e lavora a ritroso. Se è già noto che la query è vera, non occorre fare nulla. In caso contrario, l'algoritmo trova tutte le implicazioni nella base di conoscenza che hanno q come conclusione: se tutte le premesse di una di quelle implicazioni possono essere dimostrate vere mediante la concatenazione all'indietro, allora q è vera. Applicato alla query Q della Figura 7.16, l'algoritmo percorre il grafo verso il basso finché non raggiunge un insieme di fatti noti che formano la base della dimostrazione. L'algoritmo è sostanzialmente identico a quello indicato come RICERCA-AND-OR nella Figura 4.11 del Capitolo 4. Come nel caso della concatenazione in avanti, un'implementazione efficiente si esegue in tempo lineare.

La concatenazione all'indietro è una forma di **ragionamento basato sugli obiettivi**. È utile per rispondere a questioni specifiche come "Cosa devo fare adesso?" oppure "Dove avrò lasciato le chiavi?". Spesso il costo della concatenazione all'indietro è *molto meno* che lineare nelle dimensioni della base di conoscenza, perché il processo coinvolge solo i fatti rilevanti.

**ragionamento
basato sugli
obiettivi**

7.6 Model checking proposizionale efficiente

In questo paragrafo descriviamo due famiglie di algoritmi efficienti per l'inferenza proposizionale che si rifanno al model checking: il primo approccio è basato sulla ricerca con backtracking, il secondo sulla ricerca hill climbing. Questi algoritmi fanno parte della "tecnologia" del calcolo proposizionale. Se è la prima volta che leggete questo capitolo, potete scorrere velocemente questo paragrafo per ritornarvi in seguito.

Gli algoritmi che descriveremo servono a verificare la soddisficiabilità: il problema SAT (come si è visto in precedenza, per verificare la conseguenza logica, $\alpha \models \beta$, si può verificare la *insoddisficiabilità* di $\alpha \wedge \neg\beta$). Abbiamo già notato all'inizio del Paragrafo 7.5 l'analogia tra la ricerca di un modello che soddisfa una formula logica e quella di una soluzione per un problema di soddisfacimento di vincoli, per cui non è molto sorprendente che le due famiglie di algoritmi di soddisficiabilità proposizionale ricordino da vicino quelli con backtracking vi-

sti nel Paragrafo 6.3 e quelli di ricerca locale visti nel Paragrafo 6.4. Questo non toglie che siano molto importanti, dal momento che molti problemi combinatori incontrati in informatica possono essere ricondotti alla verifica di soddisfacibilità di una formula proposizionale. Ogni passo avanti nel campo degli algoritmi di soddisfacibilità ha enormi ripercussioni sulla nostra capacità di gestire la complessità in generale.

7.6.1 Un algoritmo di backtracking completo

**algoritmo
Davis–Putnam**

Il primo algoritmo che consideriamo viene spesso chiamato **algoritmo Davis–Putnam**, a causa del fondamentale articolo di Martin Davis e Hilary Putnam (1960). La nostra versione in effetti è quella descritta da Davis, Logemann e Loveland (1962), per cui useremo l'acronimo DPLL delle iniziali dei quattro autori. DPLL prende come input una formula in forma normale congiuntiva, ovvero un insieme di clausole. Come RICERCA-BACKTRACKING e TV-IMPLICA?, si tratta essenzialmente di un'enumerazione ricorsiva in profondità dei modelli possibili. Rispetto al semplice schema di TV-CONSEGUE?, DPLL apporta tre migliorie.

- *Terminazione anticipata*: l'algoritmo è in grado di determinare se la formula è vera o falsa anche con un modello parzialmente incompleto. Una clausola è vera se *qualsiasi* suo letterale è vero, anche se agli altri non è ancora stato assegnato un valore di verità; di conseguenza l'intera formula può essere giudicata vera ancor prima di completare il modello. Per esempio, la formula $(A \vee B) \wedge (A \vee C)$ è vera se A è vero, indipendentemente dai valori di B e C . In modo analogo, una formula è falsa se *una qualsiasi* delle sue clausole è falsa, il che accade quando sono falsi tutti i letterali che la compongono. Ancora una volta, questo può accadere molto tempo prima che il modello sia completo. La terminazione anticipata fa risparmiare all'algoritmo l'esame di interi sottoalberi nello spazio di ricerca.
- *Euristica del simbolo puro*: un **simbolo puro** è un simbolo che compare sempre con lo stesso “segno” in tutte le clausole. Per esempio, nelle tre clausole $(A \vee \neg B)$, $(\neg B \vee \neg C)$ e $(C \vee A)$, il simbolo A è puro perché compare solo come letterale positivo, B è puro perché compare solo il letterale negativo, e C è impuro. È facile vedere che se una formula ha un modello, i valori di verità dei simboli puri saranno assegnati in modo che i corrispondenti letterali valgano *true*, perché questo non potrà mai rendere falsa una clausola. Notate che, determinando la purezza di un simbolo, l'algoritmo può ignorare le clausole che sono già note come vere nel modello costruito fino a quel momento. Per esempio, se il modello contiene $B = \text{false}$, allora la clausola $(\neg B \vee \neg C)$ è già vera, e nelle rimanenti clausole C compare solo come letterale positivo, perciò C diventa un simbolo puro.
- *Euristica della clausola unitaria*: abbiamo già definito una **clausola unitaria** come una clausola che contiene un solo letterale. Nel contesto di DPLL, il termine indica anche clausole in cui è già stato assegnato dal modello il valore *false* a tutti i letterali tranne uno. Per esempio, se il modello contiene $B = \text{true}$, allora $(\neg B \vee \neg C)$ si semplifica in $\neg C$, che è una clausola unitaria. Palesemente, affinché questa clausola sia vera, a C dev'essere assegnato il valore *false*. L'euristica della clausola unitaria esegue tutti questi assegnamenti relativi a clausole unitarie prima di considerare i restanti assegnamenti. Una conseguenza importante di questa euristica è che ogni tentativo di dimostrare (per refutazione) un letterale che è già presente nella base di conoscenza avrà immediatamente successo (cfr. Esercizio 7.KNOW). Notate anche che assegnare un valore a una clausola unitaria può crearne un'altra: per esempio, quando si assegna a C il valore *false*, $(C \vee A)$ diventa una clausola unitaria, e questo fa sì che A riceva il valore *true*. Questa “cascata” di assegnamenti forzati è chiamata **propagazione delle unità** e ricorda il processo di concatenazione in avanti con le clausole definite. In effetti, se l'espressione CNF contiene solo clausole definite, DPLL essenzialmente non fa che replicare la concatenazione in avanti (cfr. Esercizio 7.DPLL).

simbolo puro

**propagazione
delle unità**

```

function DPLL-SODDISFACIBILE?(s) returns true oppure false
  inputs: s, una formula della logica proposizionale

    clausole  $\leftarrow$  l'insieme di clausole nella rappresentazione CNF di s
    simboli  $\leftarrow$  una lista di tutti i simboli proposizionali in s
    return DPLL(clausole, simboli, {})

function DPLL(clausole, simboli, modello) returns true oppure false
  if ogni clausola in clausole è vera in modello then return true
  if qualche clausola in clausole è falsa in modello then return false
  P, valore  $\leftarrow$  TROVA-SIMBOLO-PURO(simboli, clausole, modello)
  if P è diverso da null then return DPLL(clausole, simboli - P, modello  $\cup$  {P = valore})
  P, valore  $\leftarrow$  TROVA-CLAUSOLA-UNITARIA(clausole, modello)
  if P è diverso da null then return DPLL(clausole, simboli - P, modello  $\cup$  {P = valore})
  P  $\leftarrow$  PRIMO(simboli); resto  $\leftarrow$  RESTO(simboli)
  return DPLL(clausole, resto, modello  $\cup$  {P = true}) or
        DPLL(clausole, resto, modello  $\cup$  {P = false})

```

Figura 17.7 L'algoritmo DPLL per la verifica della soddisfabilità di una formula della logica proposizionale. Le idee alla base di TROVA-SIMBOLO-PURO e TROVA-CLAUSOLA-UNITARIA sono descritte nel testo; entrambi restituiscono un simbolo (o null) e il valore di verità da assegnare a tale simbolo. Come TV-CONSEGUE?, l'algoritmo DPLL lavora su modelli parziali.

L'algoritmo DPLL è mostrato nella Figura 7.17: abbiamo riportato solo lo schema del processo di ricerca senza i dettagli dell'implementazione.

La Figura 7.17 non mostra i trucchi che consentono ai risolutori di problemi SAT di affrontare problemi di grandi dimensioni. È interessante il fatto che la maggior parte di questi trucchi è abbastanza generale, e ne abbiamo visti alcuni precedentemente, sotto mentite spoglie.

1. **Analisi di componenti** (vista nel caso della Tasmania nei CSP): con l'assegnamento di valori di verità alle variabili da parte del DPLL, l'insieme di clausole può venire separato in sottoinsiemi disgiunti, chiamati **componenti**, che non hanno in comune variabili non assegnate. Dato un modo efficiente di rilevare quando ciò accade, un risolutore è in grado di aumentare notevolmente la velocità lavorando separatamente su ciascun componente.
2. **Ordinamento di variabili e di valori** (visto nel Paragrafo 6.3.1 per i CSP): la nostra semplice implementazione di DPLL utilizza un ordinamento di variabili arbitrario e prova sempre il valore *true* prima di *false*. **L'euristica di grado** (Paragrafo 6.3.1) suggerisce di scegliere la variabile che appare più frequentemente su tutte le clausole rimanenti.
3. **Backtracking intelligente** (visto nel Paragrafo 6.3.3 per CSP): molti problemi che non si risolvono in ore di esecuzione usando il backtracking cronologico, si risolvono in pochi secondi con il backtracking intelligente che risale fino al punto di conflitto. Tutti i risolutori SAT che eseguono il backtracking intelligente utilizzano qualche forma di **apprendimento delle clausole di conflitto** per registrare i conflitti in modo che non si ripetano nel prosieguo della ricerca. Solitamente viene mantenuto un insieme di conflitti di dimensione limitata, e quelli usati raramente sono rimossi.
4. **Riavvio casuale** (visto nel Paragrafo 4.1.1 per l'hill climbing): talvolta un'esecuzione sembra non fare progressi; in questo caso possiamo riprendere dall'inizio dell'albero di ricerca, invece che cercare di continuare. Dopo il riavvio, si fanno scelte casuali diverse (nella

selezione di variabili e valori). Le clausole apprese durante la prima esecuzione sono mantenute dopo il riavvio e possono aiutare a potare lo spazio di ricerca. Il riavvio non garantisce che si troverà più rapidamente una soluzione, ma riduce la varianza sul tempo di risoluzione.

5. **Indicizzazione intelligente** (vista in molti algoritmi): i metodi di velocizzazione utilizzati in DPLL, come i trucchi usati nei moderni risolutori, richiedono una rapida indicizzazione di elementi come “l’insieme di clausole in cui la variabile X_i appare come letterale positivo”. Questo compito è complicato dal fatto che gli algoritmi sono interessati soltanto alle clausole che non sono ancora state soddisfatte da precedenti assegnamenti alle variabili, perciò le strutture di indicizzazione devono essere aggiornate dinamicamente mentre l’elaborazione procede.

Con questi miglioramenti, i moderni risolutori sono in grado di affrontare problemi con decine di milioni di variabili. Grazie a essi sono state rivoluzionate aree quali la verifica dell’hardware e dei protocolli di sicurezza, che in precedenza richiedevano laboriose prove manuali.

7.6.2 Algoritmi di ricerca locale

Abbiamo già visto diversi algoritmi di ricerca locale, tra cui HILL-CLIMBING (Paragrafo 4.1.1) e SIMULATED-ANNEALING (Paragrafo 4.1.2). Questi algoritmi possono essere applicati direttamente ai problemi di soddisficiabilità, a patto di scegliere la giusta funzione di valutazione. Dato che l’obiettivo è trovare un assegnamento che soddisfa ogni clausola, sarà sufficiente una funzione di valutazione che conta il numero di clausole non soddisfatte. In effetti questa è esattamente la misura utilizzata dall’algoritmo MIN-CONFLICTS, utilizzato per i CSP (Paragrafo 6.4). Tutti questi algoritmi si muovono nello spazio degli assegnamenti completi, invertendo il valore di verità di un simbolo per volta. Lo spazio normalmente contiene molti minimi locali, per sfuggire ai quali si possono introdurre varie forme di perturbazione casuale. In anni recenti sono stati effettuate molte sperimentazioni per trovare un compromesso soddisfacente tra “avidità” (*greediness*) e casualità.

Uno dei più semplici ed efficaci algoritmi scaturiti da questa ricerca è WALKSAT (Figura 7.18). A ogni iterazione, l’algoritmo seleziona una clausola non soddisfatta e cambia il valore di verità di uno dei suoi simboli. Ci sono due modi per scegliere tale simbolo, e l’algoritmo

```
function WALKSAT(clausole, p, max_flips) returns un modello o fallimento
  inputs: clausole, un insieme di clausole della logica proposizionale
    p, la probabilità di effettuare una “camminata casuale”, tipicamente intorno a 0,5
    max_flips, numero massimo di inversioni di valore prima di abbandonare

  modello  $\leftarrow$  un assegnamento casuale di valori di verità ai simboli in clausole
  for i = 1 to max_flips do
    if modello soddisfa clausole then return modello
    clausola  $\leftarrow$  una clausola, falsa in modello, scelta casualmente nell’insieme clausole
    if RANDOM(0, 1) <= p then inverti il valore in modello di un simbolo scelto casualmente in clausola
    else inverti il valore di verità del simbolo in clausole che massimizza il numero di clausole soddisfatte
  return fallimento
```

Figura 7.18 L’algoritmo WALKSAT verifica la soddisficiabilità invertendo casualmente i valori delle variabili. Ne esistono diverse versioni.

utilizza l'uno o l'altro casualmente: (1) un passo “a conflitti minimi” che minimizza il numero di clausole non soddisfatte nel nuovo stato; (2) una “camminata casuale” (*random walk*) che sceglie il simbolo del tutto casualmente.

Quando WALKSAT restituisce un modello, la formula in input dev'essere necessariamente soddisfacibile, ma quando restituisce un *fallimento*, esistono due possibili cause: la formula è insoddisfacibile o si deve concedere all'algoritmo altro tempo. Se poniamo $\text{max_flips} = \infty$ e $p > 0$, WALKSAT prima o poi restituirà un modello (a patto che ne esista uno), perché i passi della camminata casuale alla fine arriveranno alla soluzione. Ma, ahinoi, se max_flips è infinito e la formula è insoddisfacibile, l'algoritmo non terminerà mai l'esecuzione!

Per questo motivo, WALKSAT è più utile quando ci aspettiamo che una soluzione esista davvero: i problemi discussi nei Capitoli 3 e 6, per esempio, solitamente hanno soluzioni. D'altra parte, la ricerca locale non può mai determinare l'*insoddisfacibilità*, che è richiesta per determinare la conseguenza logica. Per esempio, un agente non può usare la ricerca locale in modo *affidabile* per dimostrare che, in un mondo del wumpus, una particolare stanza è sicura. Al massimo sarebbe in grado di affermare: “Ci ho pensato per un'ora e non sono riuscito a trovare neppure un mondo possibile in cui quella stanza *non è sicura*”. Questo potrebbe essere un buon indicatore empirico che la stanza è sicura, ma non è certamente una dimostrazione.

7.6.3 Il panorama dei problemi SAT casuali

Alcuni problemi SAT sono più difficili di altri. I problemi *facili* possono essere risolti da qualsiasi algoritmo, ma poiché sappiamo che SAT è NP-completo, almeno alcune istanze di problemi richiederanno un tempo di esecuzione esponenziale. Nel Capitolo 6 abbiamo fatto delle scoperte sorprendenti riguardo certe categorie di problemi: per esempio quello delle n regine, che per lungo tempo è stato ritenuto particolarmente difficoltoso per gli algoritmi di ricerca con backtracking, si è rivelato banalmente semplice per i metodi di ricerca locale come min-conflicts. Questo è dovuto al fatto che le soluzioni sono distribuite densamente nello spazio degli assegnamenti, ed è garantito che ogni assegnamento iniziale abbia una soluzione vicina. Il problema a n regine è quindi facile perché è **sotto-vincolato**.

Quando consideriamo i problemi di soddisfacibilità in forma normale congiuntiva, diciamo che un problema è sotto-vincolato quando ci sono relativamente *poche* clausole che, appunto, vincolano le variabili. Per esempio, ecco una formula 3-CNF generata casualmente con cinque simboli e cinque clausole:

$$\begin{aligned} & (\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \\ & \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C). \end{aligned}$$

16 dei 32 assegnamenti possibili sono modelli della formula, per cui in media basteranno solo due tentativi casuali per trovare un modello. Questo è un problema di soddisfacibilità facile, come la maggior parte dei problemi sotto-vincolati. D'altra parte, un problema *sovra-vincolato* presenta molte clausole rispetto al numero di variabili e probabilmente non avrà soluzioni. I problemi sovra-vincolati sono spesso facili da risolvere, perché i vincoli conducono rapidamente a una soluzione oppure a un vicolo cieco senza possibilità di uscirne.

Per andare oltre queste intuizioni di base, dobbiamo definire con esattezza il modo in cui sono generate clausole casuali. La notazione $\text{CNF}_k(m, n)$ denota una formula k -CNF con m clausole e n simboli, in cui le clausole sono scelte in modo uniforme, indipendente e senza sostituzione tra tutte le clausole con k letterali diversi, che sono casualmente positivi o negativi (un simbolo non può apparire due volte in una clausola, e una clausola non può apparire due volte in una formula).

Data una sorgente di formule casuali, possiamo misurare la probabilità della soddisfacibilità. La Figura 7.19(a) traccia la probabilità di $\text{CNF}_3(m, 50)$, ovvero formule con 50 varia-

sotto-vincolato

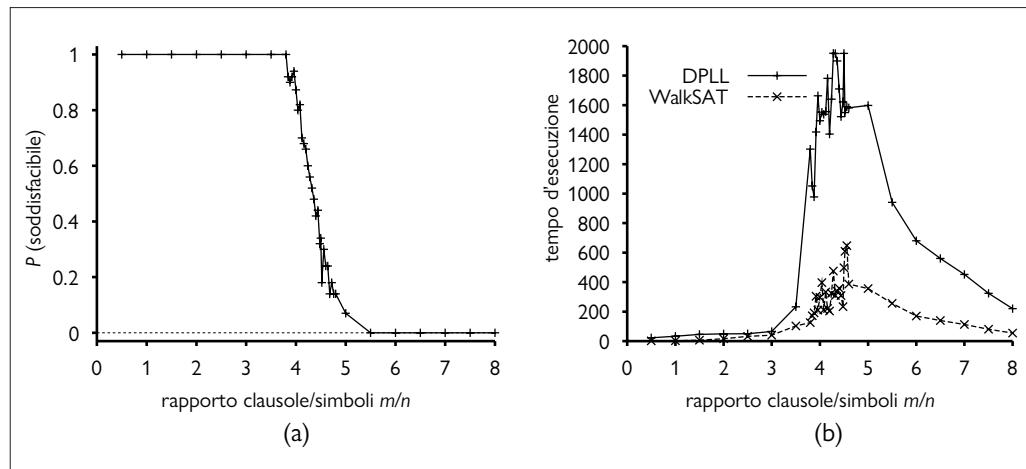


Figura 7.19 (a) Un grafico che mostra la probabilità che una formula 3-CNF casuale con $n = 50$ simboli sia soddisfacibile in funzione del rapporto clausole/simboli m/n . (b) Grafico del tempo medio di esecuzione (misurato in numero di iterazioni) per DPLL e WALKSAT su formule casuali 3-CNF. Nei problemi più difficili il rapporto clausole/simboli è di circa 4,3.

**congettura
della soglia
di soddisficiabilità**

bili e 3 letterali per clausola, in funzione del rapporto clausole/simboli m/n . Com’è prevedibile, per m/n piccoli la probabilità si avvicina a 1, mentre al crescere di m/n tende a zero. Notate che la probabilità diminuisce in modo abbastanza brusco intorno a $m/n = 4,3$. Empiricamente troviamo che la “parete verticale” è situata più o meno nello stesso punto (per $k = 3$) e diventa sempre più verticale al crescere di n . Teoricamente, la **congettura della soglia di soddisficiabilità** dice che, per ogni $k \geq 3$, esiste un rapporto di soglia r_k tale che, al tendere di n a infinito, la probabilità che $CNF_k(rn, n)$ sia soddisfacibile diventa 1 per tutti i valori di r al di sotto della soglia, e 0 per tutti i valori al di sopra. Questa congettura rimane senza dimostrazione, anche per casi speciali come $k = 3$. A prescindere dal fatto che possa essere definita un teorema o meno, questo tipo di effetto soglia è certamente comune per i problemi di soddisficiabilità ma anche per altri tipi di problemi NP-difficili.

Ora che ci siamo fatti un’idea abbastanza precisa di dove sono situati i problemi soddisfacibili e insoddisfacibili, ci chiediamo dove siano i problemi difficili. Risulta che anch’essi sono spesso nei pressi del valore soglia. La Figura 7.19(b) mostra che problemi con 50 simboli con valore soglia di 4,3 sono 20 volte più difficili da risolvere di quelli con rapporto di 3,3. I problemi sotto-vincolati sono i più facili da risolvere (perché è facile indovinare una soluzione); quelli sovra-vincolati non sono facili quanto i precedenti, ma sono sempre molto più facili di quelli situati intorno alla soglia.

7.7 Agenti basati sulla logica proposizionale

In questo paragrafo riprendiamo tutti i concetti che abbiamo visto sin qui per costruire agenti del mondo del wumpus che utilizzano la logica proposizionale. Il primo passo è consentire all’agente di dedurre, per quanto possibile, lo stato del mondo date le sue percezioni pregresse. Per fare ciò dobbiamo scrivere un modello logico completo degli effetti delle azioni. Mostriamo poi come un agente possa usare l’inferenza logica nel mondo dei wumpus. E dobbiamo anche mostrare come l’agente possa tenere traccia del mondo in modo efficiente senza ritornare alla storia delle percezioni per ogni inferenza. Infine, mostriamo come l’agente possa utilizzare l’inferenza logica per costruire piani con garanzia di raggiungere gli obiettivi, purché la sua base di conoscenza sia vera nel mondo reale.

7.7.1 Lo stato corrente del mondo

Come si è detto all'inizio di questo capitolo, un agente logico opera deducendo che cosa fare da una base di conoscenza di formule che riguardano il mondo. La base di conoscenza è composta da assiomi (conoscenza generale sul funzionamento del mondo) e formule di percezione ottenute dall'esperienza dell'agente in un mondo particolare. In questo paragrafo ci concentriamo sul problema di dedurre lo stato corrente del mondo del wumpus: dove mi trovo, se una stanza è sicura e così via.

Cominciamo raccogliendo gli assiomi del Paragrafo 7.4.3. L'agente sa che la stanza di partenza non contiene pozzi ($\neg P_{1,1}$) né wumpus ($\neg W_{1,1}$). Inoltre, per ogni stanza, l'agente sa che in essa si sente uno spostamento d'aria se e solo se una stanza adiacente contiene un pozzo; e una stanza è puzzolente se e solo se una stanza adiacente contiene un wumpus. Di conseguenza, inseriamo una grande raccolta di formule della forma seguente:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$$

...

L'agente sa anche che esiste esattamente un wumpus. Questa conoscenza viene espressa in due parti; prima di tutto dobbiamo dire che ce n'è *almeno uno*:

$$W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,3} \vee W_{4,4}.$$

Quindi dobbiamo dire che c'è *al massimo un* wumpus. Per ogni coppia di stanze, aggiungiamo una formula che dice che almeno una di esse deve essere priva di wumpus:

$$\neg W_{1,1} \vee \neg W_{1,2}$$

$$\neg W_{1,1} \vee \neg W_{1,3}$$

...

$$\neg W_{4,3} \vee \neg W_{4,4}.$$

Fin qui tutto bene. Ora consideriamo le percezioni dell'agente. Stiamo utilizzando $S_{1,1}$ per indicare che c'è fetore in [1,1]; possiamo utilizzare una singola proposizione, *Fetore*, per indicare che l'agente percepisce un fetore? Sfortunatamente non possiamo: se nel precedente passo temporale non vi era fetore, \neg *Fetore* sarebbe già asserito, e la nuova asserzione porterebbe a una contraddizione. Il problema si risolve quando ci rendiamo conto che una percezione asserisce qualcosa *soltanto in relazione al tempo corrente*. Perciò, se il passo temporale (fornito a COSTRUISCI-FORMULA-PERCEZIONE nella Figura 7.1) è 4, aggiungiamo *Fetore*⁴ alla base di conoscenza, invece di *Fetore*, evitando così qualunque contraddizione con \neg *Fetore*³. Lo stesso vale per percezioni di brezza, urto contro il muro, scintillio e ululato.

L'idea di associare proposizioni a passi temporali si estende a qualsiasi aspetto del mondo che cambia nel tempo. Per esempio, la base di conoscenza iniziale include $L_{1,1}^0$ (l'agente si trova nella stanza [1, 1] al tempo 0) oltre a *RivoltoEst*⁰, *HaFreccia*⁰ e *WumpusVivo*⁰. Utilizziamo il termine **fluente** per indicare un aspetto del mondo che cambia. "Fluente" è sinonimo di "variabile di stato", nel senso descritto nella trattazione delle rappresentazioni fatto-rizzate nel Paragrafo 2.4.7. I simboli associati ad aspetti permanenti del mondo non richiedono un indice temporale e sono talvolta chiamati **variabili atemporali**.

fluente

Possiamo collegare direttamente le percezioni di fetore e brezza alle proprietà delle stanze in cui vengono ricevute nel modo seguente:¹¹ per ogni passo temporale t e ogni stanza $[x, y]$ asseriamo:

variabile atemporale

¹¹ Nel Paragrafo 7.4.3, per comodità, si è sorvolato su questo requisito.

$$\begin{aligned} L'_{x,y} &\Rightarrow (Brezza' \Leftrightarrow B_{x,y}) \\ L'_{x,y} &\Rightarrow (Fetore' \Leftrightarrow S_{x,y}). \end{aligned}$$

Ora naturalmente abbiamo bisogno di assiomi che consentano all’agente di tenere traccia di fluenti quali $L'_{x,y}$. Questi fluenti cambiano a seguito di azioni compiute dall’agente, perciò nella terminologia del Capitolo 3 dobbiamo scrivere il **modello di transizione** del mondo del wumpus come insieme di formule logiche.

Per prima cosa ci servono dei simboli proposizionali per le occorrenze di azioni. Come per le percezioni, questi simboli sono indicizzati in riferimento al tempo: *Avanti*⁰ significa che l’agente esegue l’azione *Avanti* al tempo 0. Per convenzione, la percezione corrispondente a un dato passo temporale si verifica per prima, seguita poi dall’azione corrispondente allo stesso passo, seguita da una transizione al passo temporale successivo.

assiomi da effetto

Per descrivere come cambia il mondo, possiamo provare a scrivere **assiomi di effetto** che specificano il risultato di un’azione al successivo passo temporale. Per esempio, se l’agente si trova nella posizione [1, 1] rivolto a est al tempo 0 e va *Avanti*, il risultato è che si trova nella stanza [2, 1] e non più in [1, 1]:

$$L^0_{1,1} \wedge RivoltoEst^0 \wedge Avanti^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1}). \quad (7.1)$$

Ci serve una formula simile per ogni possibile passo temporale, per ognuna delle 16 stanze e per ognuno dei quattro orientamenti. Ci servirebbero formule simili anche per le altre azioni: *Afferra*, *Scocca*, *Esci*, *GiraSinistra* e *GiraDestra*.

Supponiamo che l’agente decida di muoversi *Avanti* al tempo 0 e asserisca questo fatto nella propria base di conoscenza. Dato l’assioma di effetto dell’Equazione (7.1), insieme alle asserzioni iniziali relative allo stato al tempo 0, l’agente può dedurre che si trova in [2, 1]. Ovvero, $\text{ASK}(KB, L^1_{2,1}) = \text{true}$.

Fin qui tutto bene. Sfortunatamente, se chiediamo $\text{ASK}(KB, HaFreccia^1)$, la risposta è *false*, cioè l’agente non è in grado di provare che ha ancora la freccia, né che *non ce l’ha!* L’informazione è andata persa perché l’assioma di effetto non è in grado di stabilire che cosa rimane *invariato* a seguito di un’azione. La necessità di fare ciò porta al **problema del frame**.¹² Una possibile soluzione di questo problema consisterebbe nell’aggiungere **assiomi di frame** che asseriscano esplicitamente tutte le proposizioni che rimangono invariate. Per esempio, per ogni tempo t avremmo:

$$\begin{aligned} Avanti^t &\Rightarrow (HaFreccia' \Leftrightarrow HaFreccia^{t+1}) \\ Avanti^t &\Rightarrow (WumpusVivo' \Leftrightarrow WumpusVivo^{t+1}) \\ &\dots \end{aligned}$$

dove menzioniamo esplicitamente ogni proposizione che rimane invariata dal tempo t al tempo $t + 1$ sotto l’azione *Avanti*. Ora l’agente sa che ha ancora la freccia dopo lo spostamento in avanti e che il wumpus non è morto o ritornato in vita, tuttavia la proliferazione di assiomi di frame sembra davvero inefficiente. In un mondo con m diverse azioni e n fluenti, l’insieme di assiomi del frame avrebbe dimensione $O(mn)$. Questa specifica manifestazione del problema del frame è talvolta chiamata **problema di rappresentazione del frame**

problema del frame assioma di frame

problema di rappresentazione del frame

¹² Il termine “problema del frame” deriva da “frame di riferimento” in fisica (lo sfondo assunto come stazionario rispetto al quale si misura il movimento). C’è anche un’analogia con i frame o fotogrammi di un film, in cui normalmente la maggior parte dello sfondo rimane inalterata, mentre i cambiamenti si verificano in primo piano.

e si tratta di un problema significativo, storicamente, per gli studiosi di IA; lo esamineremo ulteriormente nelle note riportate al termine di questo capitolo.

Il problema di rappresentazione del frame è significativo perché il mondo reale ha moltissimi fluenti, a dire poco. Fortunatamente per noi umani, ogni azione generalmente modifica solo un piccolo numero k di fluenti: il mondo esibisce **località**. Per risolvere il problema di rappresentazione del frame è necessario definire il modello di transizione con un insieme di assiomi di dimensione $O(mk)$ anziché $O(mn)$. Esiste anche un **problema inferenziale del frame**: il problema di proiettare in avanti i risultati di un piano di azione di t passi in tempo $O(kt)$ anziché $O(nt)$.

La soluzione di questo problema richiede di concentrarsi non sulla scrittura di assiomi relativi alle *azioni*, ma sulla scrittura di assiomi relativi ai *fluenti*. Quindi, per ogni fluente F avremo un assioma che definisce il valore di verità di F^{t+1} in termini di fluenti (incluso F) al tempo t e delle azioni che potrebbero essersi verificate al tempo t . Ora, il valore di verità di F^{t+1} può essere impostato in due modi: o l'azione al tempo t rende F vero in $t+1$, oppure F era già vero al tempo t e l'azione al tempo t non lo rende falso. Un assioma di questa forma è chiamato **assioma di stato successore** e ha il seguente schema:

$$F^{t+1} \Leftrightarrow \text{AzioneCausa}F^t \vee (F^t \wedge \neg \text{AzioneCausa} \neg F^t).$$

Uno dei più semplici assiomi di stato successore è quello di *HaFreccia*. Poiché non c'è un'azione per il ricaricamento, la parte *AzioneCausa* F^t scompare e rimane:

$$\text{HaFreccia}^{t+1} \Leftrightarrow (\text{HaFreccia}^t \wedge \neg \text{Scocca}^t). \quad (7.2)$$

Per quanto riguarda la posizione dell'agente, gli assiomi di stato successore sono più elaborati. Per esempio, $L_{1,1}^{t+1}$ è vero se (a) l'agente si è mosso *Avanti* da [1, 2] quando era rivolto a sud, o da [2, 1] quando era rivolto a ovest, oppure (b) $L_{1,1}^t$ era già vero e l'azione non ha causato movimento (perché non era *Avanti*, o perché ha portato a sbattere contro un muro). Nel formalismo della logica proposizionale, tutto ciò diventa:

$$\begin{aligned} L_{1,1}^{t+1} \Leftrightarrow & (L_{1,1}^t \wedge (\neg \text{Avanti}^t \vee \text{Urto}^{t+1})) \\ & \vee (L_{1,2}^t \wedge (\text{RivoltoSud}^t \wedge \text{Avanti}^t)) \\ & \vee (L_{2,1}^t \wedge (\text{RivoltoOvest}^t \wedge \text{Avanti}^t)). \end{aligned} \quad (7.3)$$

L'Esercizio 7.SSAX chiede di scrivere gli assiomi per gli altri fluenti del mondo del wumpus.

Dato un insieme completo di assiomi di stato successore e gli altri assiomi elencati all'inizio di questo paragrafo, l'agente sarà in grado di chiedere (ASK) e rispondere a qualsiasi domanda riguardo lo stato corrente del mondo. Per esempio, nel Paragrafo 7.2 la sequenza iniziale di percezioni e azioni è:

$$\begin{aligned} & \neg \text{Fetore}^0 \wedge \neg \text{Brezza}^0 \wedge \neg \text{Scintillio}^0 \wedge \neg \text{Urto}^0 \wedge \neg \text{Ululato}^0 ; \text{Avanti}^0 \\ & \neg \text{Fetore}^1 \wedge \text{Brezza}^1 \wedge \neg \text{Scintillio}^1 \wedge \neg \text{Urto}^1 \wedge \neg \text{Ululato}^1 ; \text{GiraDestra}^1 \\ & \neg \text{Fetore}^2 \wedge \text{Brezza}^2 \wedge \neg \text{Scintillio}^2 \wedge \neg \text{Urto}^2 \wedge \neg \text{Ululato}^2 ; \text{GiraDestra}^2 \\ & \neg \text{Fetore}^3 \wedge \text{Brezza}^3 \wedge \neg \text{Scintillio}^3 \wedge \neg \text{Urto}^3 \wedge \neg \text{Ululato}^3 ; \text{Avanti}^3 \\ & \neg \text{Fetore}^4 \wedge \neg \text{Brezza}^4 \wedge \neg \text{Scintillio}^4 \wedge \neg \text{Urto}^4 \wedge \neg \text{Ululato}^4 ; \text{GiraDestra}^4 \\ & \neg \text{Fetore}^5 \wedge \neg \text{Brezza}^5 \wedge \neg \text{Scintillio}^5 \wedge \neg \text{Urto}^5 \wedge \neg \text{Ululato}^5 ; \text{Avanti}^5 \\ & \text{Fetore}^6 \wedge \neg \text{Brezza}^6 \wedge \neg \text{Scintillio}^6 \wedge \neg \text{Urto}^6 \wedge \neg \text{Ululato}^6 \end{aligned}$$

A questo punto abbiamo $\text{ASK}(KB, L_{1,2}^6) = \text{true}$, perciò l'agente sa dove si trova. Inoltre, $\text{ASK}(KB, W_{1,3}) = \text{true}$ e $\text{ASK}(KB, P_{3,1}) = \text{true}$, perciò l'agente ha trovato il wumpus e uno dei pozzi. La domanda più importante, per l'agente, è se sia sicuro (*OK*) entrare in una stanza, cioè se la stanza non contenga pozzi o un wumpus vivo. È utile aggiungere assiomi corrispondenti, della forma:

$$OK'_{x,y} \Leftrightarrow \neg P_{x,y} \wedge \neg (W_{x,y} \wedge \text{WumpusVivo}^t).$$

località

problema inferenziale del frame

assioma di stato successore

**problema
della qualificazione**

agente ibrido

Infine, $\text{ASK}(KB, OK_{2,2}^6) = \text{true}$, perciò si può entrare nella stanza [2, 2]. In effetti, dato un algoritmo di inferenza corretto e completo quale DPLL, l’agente può rispondere a qualsiasi domanda a cui sia possibile rispondere riguardo quali stanze siano OK, e può farlo in pochi millisecondi, per mondi del wumpus di dimensioni piccole o medie.

La risoluzione dei problemi di rappresentazione e inferenziale del frame è un notevole passo avanti, ma rimane un problema serio: dobbiamo confermare che *tutte* le precondizioni necessarie di un’azione valgano, perché questa ottenga l’effetto desiderato. Abbiamo detto che l’azione *Avanti* muove l’agente in avanti a meno che non vi sia un muro lungo il percorso, ma ci sono molte altre eccezioni insolite che potrebbero causare il fallimento dell’azione: l’agente potrebbe inciampare e cadere, essere colpito da un attacco di cuore, essere portato via da pipistrelli giganti e così via. Specificare tutte queste eccezioni è il **problema della qualificazione**. Non esiste una soluzione completa, rimanendo nell’ambito della logica; i progettisti del sistema devono usare il buon senso per decidere il livello di dettaglio desiderato nello specificare il loro modello, e quali dettagli lasciar fuori. Nel Capitolo 12 vedremo che la teoria della probabilità ci consente di considerare tutte le eccezioni senza nominarle esplicitamente.

7.7.2 Un agente ibrido

La capacità di dedurre vari aspetti dello stato del mondo può essere combinata in modo abbastanza semplice con regole condizione-azione (cfr. Paragrafo 2.4.2) e con gli algoritmi di risoluzione dei problemi trattati nei Capitoli 3 e 4 per produrre un **agente ibrido** per il mondo del wumpus. La Figura 7.20 mostra un possibile modo per farlo. Il programma agente mantiene e aggiorna una base di conoscenza e un piano corrente. La base di conoscenza iniziale contiene gli assiomi *atemporali*, quelli che non dipendono da t , quali l’assioma che mette in relazione la percezione di spostamenti d’aria nelle stanze alla presenza di pozzi. A ogni passo temporale, la nuova formula di percezione viene aggiunta insieme a tutti gli assiomi che dipendono da t , come quelli di stato successore (nel paragrafo seguente spiegheremo perché l’agente non necessita di assiomi per passi temporali *futuri*). Poi l’agente utilizza l’inferenza logica, interrogando la base di conoscenza (ASK) per determinare quali stanze sono sicure e quali devono ancora essere visitate.

Il corpo principale del programma agente costruisce un piano basato su una priorità decrescente di obiettivi. Per prima cosa, se c’è uno scintillio, il programma costruisce un piano per afferrare l’oro, seguire un percorso di ritorno alla posizione iniziale e arrampicarsi fuori dalla caverna. Altrimenti, se non c’è un piano corrente, il programma pianifica un percorso verso la stanza sicura più vicina che non è ancora stata visitata, assicurandosi che il percorso attraversi soltanto stanze sicure. La pianificazione del percorso è svolta con la ricerca A*, non con ASK. Se non ci sono stanze sicure da esplorare, il passo successivo – se l’agente ha ancora una freccia – è quello di tentare di rendere sicura una stanza scoccando una freccia in una delle possibili posizioni del wumpus. Tali posizioni sono determinate chiedendo dove $\text{ASK}(KB, \neg W_{x,y})$ è falso, cioè dove non è noto che non c’è un wumpus. La funzione PIANIFICA-SCOCCA (che non mostriamo) utilizza PIANIFICA-PERCORSO per pianificare una sequenza di azioni che effettuerà il puntamento per questo colpo. Se fallisce, il programma cerca una stanza da esplorare che non sia provatamente insicura, ovvero una stanza per cui $\text{ASK}(KB, OK'_{x,y})$ è falso. Se una tale stanza non esiste, allora la missione è impossibile e l’agente ripiega in [1, 1] ed esce dalla caverna.

7.7.3 Stima dello stato con la logica

Il programma agente della Figura 7.20 funziona abbastanza bene, ma ha un importante punto debole: con lo scorrere del tempo, il costo computazionale per le chiamate di ASK continua ad aumentare. Ciò accade principalmente perché le inferenze richieste devono tornare sempre più indietro nel tempo e coinvolgono un numero sempre maggiore di simboli pro-

```

function AGENTE-WUMPUS-IBRIDO(percezione) returns un'azione
  inputs: percezione, una lista, [fetore,brezza,scintillio,urto,ululato]]
  persistent: KB, una base di conoscenza, che inizialmente contiene le caratteristiche atemporali della
    "fisica del wumpus".
    t, un contatore, inizialmente 0, che indica il tempo
    piano, una sequenza di azioni, inizialmente vuota
  TELL(KB,COSTRUISCI-FORMULA-PERCEZIONE(percezione, t))
  inserisci, con TELL, nella KB le formule temporali della "fisica" per il tempo t
  sicuro  $\leftarrow \{[x, y] : \text{ASK}(KB, OK_{x,y}^t) = \text{true}\}$ 
  if ASK(KB,Scintilliot) = true then
    piano  $\leftarrow [\text{Afferra}] + \text{PIANIFICA-PERCORSO}(\text{corrente}, \{[1,1]\}, \text{sicuro}) + [\text{Esci}]$ 
  if piano è vuoto then
    nonvisitate  $\leftarrow \{[x, y] : \text{ASK}(KB, L_{x,y}^{t'}) = \text{false} \text{ per ogni } t' \leq t\}$ 
    piano  $\leftarrow \text{PIANIFICA-PERCORSO}(\text{corrente}, \text{nonvisitate} \cap \text{sicuro}, \text{sicuro})$ 
  if piano è vuoto e ASK(KB,HaFrecciat) = true then
    possibile_wumpus  $\leftarrow \{[x, y] : \text{ASK}(KB, \neg W_{x,y}) = \text{false}\}$ 
    piano  $\leftarrow \text{PIANIFICA-SCOCCA}(\text{corrente}, \text{possibile_wumpus}, \text{sicuro})$ 
  if piano è vuoto then // non ha altra scelta che prendere un rischio
    non_insicuro  $\leftarrow \{[x, y] : \text{ASK}(KB, \neg OK_{x,y}^t) = \text{false}\}$ 
    piano  $\leftarrow \text{PIANIFICA-PERCORSO}(\text{corrente}, \text{nonvisitate} \cap \text{non_insicuro}, \text{sicuro})$ 
  if piano è vuoto then
    piano  $\leftarrow \text{PIANIFICA-PERCORSO}(\text{corrente}, \{[1, 1]\}, \text{sicuro}) + [\text{Esci}]$ 
  azione  $\leftarrow \text{POP}(\text{piano})$ 
  TELL(KB,COSTRUISCI-FORMULA-AZIONE(azione, t))
  t  $\leftarrow t + 1$ 
  return azione

```

```

function PIANIFICA-PERCORSO(corrente,obiettivi,consentito) returns una sequenza di azioni
  inputs: corrente, la posizione corrente dell'agente
          obiettivi, un insieme di riquadri; tenta di pianificare un percorso verso uno di essi
          consentito, un insieme di riquadri che possono far parte del percorso

```

```

problema  $\leftarrow \text{PROBLEMA-PERCORSO}(\text{corrente}, \text{obiettivi}, \text{consentito})$ 
return RICERCA(problema) // Qualsiasi algoritmo di ricerca del Capitolo 3

```

Figura 7.20 Un programma agente ibrido per il mondo del wumpus. Utilizza una base di conoscenza proposizionale per inferire lo stato del mondo e una combinazione di ricerca e codice specifico del dominio per scegliere le azioni. A ogni chiamata di AGENTE-WUMPUS-IBRIDO, la funzione aggiunge una percezione alla base di conoscenza e poi utilizza un piano definito in precedenza o ne costruisce uno nuovo ed estrae il primo passo del piano come prossima azione da svolgere.

posizionali. Ovviamente la situazione è insostenibile: non possiamo avere un agente per cui il tempo per elaborare ciascuna percezione aumenta proporzionalmente alla lunghezza della sua vita! Ci serve un tempo di aggiornamento *costante*, ovvero indipendente da *t*. La soluzione ovvia è quella di salvare, o memorizzare in una **cache**, i risultati dell'inferenza, in modo che il processo di inferenza al passo temporale successivo possa basarsi sui risultati dei passi precedenti, invece di ricominciare tutto da capo.

Come abbiamo visto nel Paragrafo 4.4, la storia delle percezioni e di tutte le loro ramificazioni può essere sostituita dallo **stato-credenza**, ovvero da una rappresentazione dell’insieme di tutti i possibili stati correnti del mondo.¹³ Il processo di aggiornamento dello stato-credenza quando si ricevono nuove percezioni si chiama **stima dello stato** (Paragrafo 4.4.4). Mentre nel Paragrafo 4.4 lo stato-credenza era un elenco esplicito di stati, qui possiamo utilizzare una formula logica che coinvolge i simboli proposizionali associati al passo temporale corrente, oltre ai simboli atemporali. Per esempio, la formula logica:

$$WumpusVivo^1 \wedge L_{2,1}^1 \wedge B_{2,1} \vee (P_{3,1} \vee P_{2,2}) \quad (7.4)$$

rappresenta l’insieme di tutti gli stati al tempo 1 in cui il wumpus è vivo, l’agente si trova in [2, 1], nella stanza si percepisce uno spostamento d’aria e c’è un pozzo in [3, 1], [2, 2] o entrambi.

Mantenere uno stato-credenza esatto nella forma di formula logica non è facile. Se ci sono n simboli fluenti per il tempo t , ci sono 2^n stati possibili, cioè assegnamenti di valori di verità a questi simboli. L’insieme degli stati-credenza è l’insieme potenza (insieme di tutti i sottinsiemi) dell’insieme degli stati fisici. Ci sono 2^n stati fisici, quindi 2^{2^n} stati-credenza. Anche se utilizzassimo la codifica più compatta possibile per le formule logiche, con ogni stato-credenza rappresentato da un numero binario univoco, ci servirebbero numeri con $\log_2(2^{2^n}) = 2^n$ bit per etichettare lo stato-credenza corrente. Ciò significa che la stima esatta dello stato potrebbe richiedere formule logiche di dimensione esponenziale nel numero di simboli.

Uno schema molto comune e naturale per la stima *approssimata* dello stato consiste nel rappresentare gli stati-credenza come congiunzioni di letterali, cioè formule 1-CNF. Per fare ciò, il programma agente prova semplicemente a dimostrare X^t e $\neg X^t$ per ogni simbolo X^t (e per ogni simbolo atemporale il cui valore di verità non è ancora noto), dato lo stato-credenza in $t - 1$. La congiunzione di letterali dimostrabili diventa il nuovo stato-credenza, mentre lo stato-credenza precedente viene scartato.

È importante comprendere che questo schema potrebbe causare la perdita di alcune informazioni con il procedere del tempo. Per esempio, se la formula nell’Equazione (7.4) fosse il vero stato-credenza, allora né $P_{3,1}$ né $P_{2,2}$ sarebbero dimostrabili singolarmente e nessuno dei due apparirebbe nello stato-credenza 1-CNF (l’Esercizio 7.HYBR esamina una possibile soluzione di questo problema). D’altra parte, poiché ogni letterale dello stato-credenza 1-CNF è dimostrato dallo stato-credenza precedente, e lo stato-credenza iniziale è un’asserzione vera, sappiamo che l’intero stato-credenza 1-CNF deve essere vero. Quindi, *l’insieme dei possibili stati rappresentati dallo stato-credenza 1-CNF include tutti gli stati che sono in effetti possibili data la storia completa delle percezioni*. Come si vede nella Figura 7.21, lo stato-credenza 1-CNF opera come semplice involucro esterno, o **approssimazione conservativa**, che racchiude lo stato-credenza esatto. Questo concetto di approssimazione conservativa per un insieme complesso è un tema ricorrente in molti campi dell’intelligenza artificiale.

approssimazione conservativa

7.7.4 Costruzione di piani mediante inferenza proposizionale

L’agente della Figura 7.20 utilizza l’inferenza logica per determinare quali stanze sono sicure, ma costruisce i piani utilizzando la ricerca A*. In questo paragrafo mostriamo come costruire piani mediante inferenza logica. L’idea di base è molto semplice:

1. Costruire una formula che includa:
 - (a) $Init^0$, una raccolta di asserzioni sullo stato iniziale;

¹³ Possiamo pensare alla stessa cronologia delle percezioni come a una rappresentazione dello stato-credenza, che però rende l’inferenza sempre più costosa al crescere della lunghezza della cronologia.

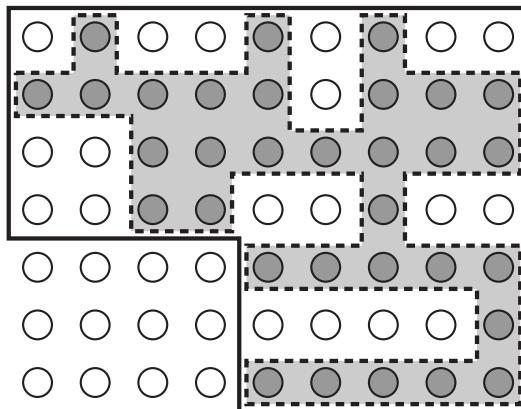


Figura 7.21 Illustrazione di uno stato-credenza 1-CNF (contorno spesso) come approssimazione conservativa, facilmente rappresentabile, dello stato-credenza esatto (area in grigio con contorno tratteggiato). Ogni mondo possibile è mostrato come un cerchio; i cerchi in grigio sono consistenti con tutte le percezioni.

```

function SATPLAN(init, transizione, obiettivo,  $T_{\max}$ ) returns soluzione o fallimento
  inputs: init, transizione, obiettivo, costituiscono una descrizione del problema
   $T_{\max}$ , un limite superiore per la lunghezza del piano

  for  $t = 0$  to  $T_{\max}$  do
    cnf  $\leftarrow$  TRADUCI-IN-SAT(init, transizione, obiettivo,  $t$ )
    modello  $\leftarrow$  RISOLTORE-SAT(cnf)
    if modello non è null then
      return ESTRAI-SOLUZIONE(modello)
    return fallimento
  
```

Figura 7.22 L'algoritmo SATPLAN. Il problema di pianificazione è tradotto in una formula CNF in cui l'obiettivo è asserito a un passo temporale fissato t , e sono inclusi assiomi per ogni passo temporale fino a t . Se l'algoritmo di soddisfacibilità trova un modello, si estrae un piano esaminando i simboli proposizionali che fanno riferimento ad azioni e che sono true nel modello. Se non esiste un modello, il processo viene ripetuto con l'obiettivo spostato un passo avanti nel tempo.

- (b) *Transizione*¹, ..., *Transizione* ^{t} , gli assiomi di stato successore per tutte le possibili azioni a ogni passo temporale, fino a un tempo massimo t ;
 - (c) l'asserzione che l'obiettivo è raggiunto al tempo t : *HaOro* ^{t} \wedge *Uscito* ^{t} .
2. Presentare l'intera formula a un risolutore SAT. Se questo trova un modello che la soddisfa, allora l'obiettivo è raggiungibile; se la formula è insoddisfacibile, allora il problema è irrisolvibile.
 3. Nel caso sia stato trovato un modello, estrarne le variabili che rappresentano azioni e che hanno assegnato *true*. Insieme, queste azioni rappresentano un piano per raggiungere gli obiettivi.

La Figura 7.22 mostra una procedura di pianificazione proposizionale, SATPLAN, che implementa l'idea di base appena presentata, con una variante. Poiché l'agente non sa quanti passi gli serviranno per raggiungere l'obiettivo, l'algoritmo tenta ogni numero possibile di passi t , fino a un valore massimo concepibile per la lunghezza del piano, T_{\max} . In questo mo-

do si ha la garanzia di trovare il piano più breve, se ne esiste uno. A causa del modo in cui SATPLAN ricerca una soluzione, questo approccio non può essere sfruttato in un ambiente parzialmente osservabile; SATPLAN si limiterebbe a impostare le variabili inosservabili ai valori che gli servono per creare una soluzione.

Il passaggio fondamentale nell'utilizzo di SATPLAN è la costruzione della base di conoscenza. A un'osservazione non attenta potrebbe sembrare che gli assiomi del mondo del wumpus descritti nel Paragrafo 7.7.1 siano sufficienti per i passaggi 1(a) e 1(b) appena descritti, tuttavia vi è una significativa differenza tra i requisiti per la conseguenza logica (verificati da ASK) e quelli per la soddisfabilità. Consideriamo per esempio la posizione dell'agente, inizialmente [1, 1], e supponiamo che l'obiettivo poco ambizioso dell'agente sia di trovarsi in [2, 1] al tempo 1. La base di conoscenza iniziale contiene $L_{1,1}^0$ e l'obiettivo è $L_{2,1}^1$. Utilizzando ASK possiamo dimostrare $L_{2,1}^1$ se è asserito *Avanti*⁰ e, a rassicurarsi, non possiamo invece dimostrare $L_{2,1}^1$ se, per esempio, è asserito *Scocca*⁰. Ora SATPLAN troverà il piano [*Avanti*⁰], e fin qui tutto bene. Sfortunatamente SATPLAN troverà anche il piano [*Scocca*⁰]. Come può essere? Per capirlo, esaminiamo il modello costruito da SATPLAN: include l'assegnamento $L_{2,1}^0$, cioè l'agente può trovarsi in [2, 1] al tempo 1 se si trova lì al tempo 0 e scocca una freccia. A questo punto ci si potrebbe chiedere: “Ma non avevamo detto che l'agente era in [1, 1] al tempo 0?” In effetti è così, ma non abbiamo detto all'agente che non poteva trovarsi in due posti contemporaneamente! Per la conseguenza logica, $L_{2,1}^0$ è sconosciuto e quindi non può essere utilizzato in una dimostrazione; per la soddisfabilità, d'altra parte, $L_{2,1}^0$ è sconosciuto e può quindi essere impostato a qualsiasi valore sia utile per rendere vero l'obiettivo.

SATPLAN è un buon strumento di debugging per le basi di conoscenza, dato che mette in evidenza punti in cui manca la conoscenza. In questo caso particolare, possiamo correggere la base di conoscenza asserendo che, a ogni passo temporale, l'agente si trova in una e una sola posizione, utilizzando una raccolta di formule simili a quelle utilizzate per asserire l'esistenza di uno e un solo wumpus. In alternativa possiamo asserire $\neg L_{x,y}^0$ per tutte le posizioni diverse da [1, 1]: l'assioma di stato successore per la posizione si occupa dei passi temporali successivi. Le stesse correzioni possono essere usate anche per assicurarsi che l'agente abbia uno e un solo orientamento per volta.

SATPLAN ha ancora in serbo delle sorprese. La prima è che trova modelli con azioni impossibili, quali scoccare senza freccia. Per capirne il motivo, dobbiamo osservare con maggiore attenzione ciò che dicono gli assiomi di stato successore (come l'Equazione (7.3)) riguardo azioni con precondizioni non soddisfatte. Tali assiomi *prevedono* correttamente che non accadrà nulla con l'esecuzione di un'azione di questo tipo (cfr. Esercizio 10.SATP), ma *non* dicono che l'azione non possa essere eseguita! Per evitare di generare piani con azioni illecite, dobbiamo aggiungere **assiomi di precondizione** che affermino che il verificarsi di un'azione richiede che le precondizioni siano soddisfatte.¹⁴ Per esempio, dobbiamo dire, per ogni tempo t , che:

$$\text{Scocca}^t \Rightarrow \text{HaFreccia}^t.$$

Questo garantisce che, se un piano seleziona l'azione *Scocca* in qualsiasi momento, l'agente in quello stesso momento deve avere una freccia.

La seconda sorpresa di SATPLAN è la creazione di piani con più azioni simultanee. Per esempio, potrebbe costruire un modello in cui *Avanti*⁰ e *Scocca*⁰ sono entrambe vere, cosa non ammessa. Per eliminare questo problema introduciamo gli **assiomi di esclusione di azioni**: per ogni coppia di azioni A_i^t e A_j^t , aggiungiamo l'assioma:

$$\neg A_i^t \vee \neg A_j^t.$$

**assioma
di precondizione**

**assioma
di esclusione
di azioni**

¹⁴ Notate che l'aggiunta di assiomi di precondizione significa che non abbiamo la necessità di includere precondizioni per le azioni negli assiomi di stato successore.

Si potrebbe puntuallizzare il fatto che muovere in avanti e scoccare una freccia nello stesso tempo non è poi così difficile, mentre, per esempio, scoccare una freccia e afferrare qualcosa è praticamente impossibile. Imponendo assiomi di esclusione di azioni soltanto su coppie di azioni che interferiscono realmente tra loro, possiamo consentire che esistano piani contenenti più azioni simultanee, e poiché SATPLAN trova il piano lecito più breve, siamo certi che trarrà vantaggio da questa caratteristica.

Per riassumere, SATPLAN trova modelli per una formula contenente lo stato iniziale, l'obiettivo, gli assiomi di stato successore, gli assiomi di precondizione e gli assiomi di esclusione di azioni. Si può mostrare che questa raccolta di assiomi è sufficiente, nel senso che non ci sono più "soluzioni" spurie. Qualunque modello che soddisfi la formula proposizionale sarà un piano valido per il problema originale. La moderna tecnologia di risoluzione di problemi SAT rende abbastanza pratico questo approccio. Per esempio, un risolutore tipo DPLL non ha difficoltà a generare la soluzione per l'istanza di mondo del wumpus illustrata nella Figura 7.2.

In questo paragrafo abbiamo descritto un approccio dichiarativo alla costruzione di agenti: l'agente opera combinando l'asserzione di formule nella base di conoscenza e l'inferenza logica. Questo approccio presenta dei punti deboli nascosti in frasi come "per ogni tempo t " e "per ogni stanza $[x, y]$ ". Per ogni agente nella pratica, queste frasi devono essere implementate da codice che genera automaticamente istanze dello schema di formula generale da inserire nella base di conoscenza. Per un mondo del wumpus di dimensione ragionevole, paragonabile a quella di un piccolo gioco per computer, potremmo avere bisogno di una griglia 100×100 e di 1000 passi temporali, che porterebbero a basi di conoscenza con decine o centinaia di milioni di formule. Questo è impraticabile e inoltre evidenzia un problema più profondo: sappiamo qualcosa del mondo del wumpus (che la "fisica" funziona nello stesso modo in tutte le stanze e in tutti i passi temporali) che non siamo in grado di esprimere direttamente nel linguaggio della logica proposizionale. Per risolvere questo problema ci serve un linguaggio più espressivo, in cui frasi come "per ogni tempo t " e "per ogni stanza $[x, y]$ " possano essere scritte in modo naturale. Questo linguaggio è la logica del primo ordine, descritta nel Capitolo 8; in essa, un mondo del wumpus di qualsiasi dimensione e durata può essere scritto in circa 10 formule logiche anziché 10 milioni o 10 mila miliardi.

7.8 Riepilogo

Abbiamo presentato gli agenti basati sulla conoscenza e mostrato come definire una logica utilizzabile per ragionare sul mondo. I punti principali sono i seguenti.

- Gli agenti intelligenti devono possedere conoscenza del mondo per formulare buone decisioni.
- All'interno degli agenti la conoscenza è rappresentata mediante **formule** espresse in un **linguaggio di rappresentazione della conoscenza** e memorizzate in una **base di conoscenza**.
- Un agente è composto da una base di conoscenza e un meccanismo inferenziale. Le formule sono memorizzate nella base di conoscenza e il meccanismo di inferenza serve a dedurre nuove formule, in base alle quali l'agente può decidere quali azioni intraprendere.
- Un linguaggio di rappresentazione è definito dalla sua **sintassi**, che specifica la struttura delle formule, e la sua **semantica**, che definisce il **valore di verità** di ogni formula in ogni **mondo possibile o modello**.
- La relazione di **conseguenza logica** tra formule è cruciale per la nostra comprensione del ragionamento. Da una formula α segue logicamente un'altra formula β se β è vera in tutti i mondi in cui α lo è. Definizioni equivalenti includono la **validità** della formula $\alpha \Rightarrow \beta$ e la **insoddisfacibilità** della formula $\alpha \wedge \neg\beta$.

- L’inferenza è il processo mediante il quale, partendo da un insieme di formule, se ne derivano di nuove. Algoritmi di inferenza **corretti** derivano *solo* le formule che sono conseguenze logiche; quelli **completi** derivano *tutte* le formule che sono conseguenze logiche.
- La **logica proposizionale** è un linguaggio semplice che consiste di **simboli proposizionali** e **connettivi logici**. Può gestire proposizioni che sono note come vere, note come false o indefinite.
- Dato un vocabolario finito di proposizioni, l’insieme dei possibili modelli è finito, per cui la conseguenza logica può essere verificata enumerando i modelli stessi. Algoritmi di **model checking** efficienti per la logica proposizionale includono metodi basati su backtracking e ricerca locale e spesso possono risolvere velocemente problemi di grandi dimensioni.
- Le **regole di inferenza** sono schemi di ragionamento corretto usati per trovare dimostrazioni. La regola di **risoluzione** è alla base di un algoritmo di inferenza completo, applicabile a basi di conoscenza in **forma normale congiuntiva**. La **concatenazione in avanti** e la **concatenazione all’indietro** sono algoritmi di ragionamento molto intuitivi, applicabili a basi di conoscenza espresse nella **forma di Horn**.
- I metodi di **ricerca locale** come WALKSAT possono essere utilizzati per trovare soluzioni. Tali algoritmi sono corretti ma non completi.
- La **stima dello stato** usando la logica richiede di mantenere una formula logica che descrive l’insieme dei possibili stati consistenti con la storia delle osservazioni. Ogni passo di aggiornamento richiede di effettuare un’inferenza utilizzando il modello di transizione dell’ambiente, che è costruito da **assiomi di stato successore** che specificano come cambia ogni **fluente**.
- In un agente logico la risoluzione SAT può essere usata per prendere decisioni: trovare modelli possibili che specificano sequenze di azioni future che raggiungono l’obiettivo. Questo approccio funziona soltanto per ambienti completamente osservabili o senza sensori.
- La logica proposizionale non è adatta ad ambienti di dimensione illimitata perché manca della potenza espressiva per affrontare in modo conciso tempo, spazio e schemi universali di relazioni tra oggetti.

Note storiche e bibliografiche

L’articolo di John McCarthy “Programs with Common Sense” (McCarthy, 1958, 1968) propose la nozione di agenti che usano il ragionamento logico per collegare percezioni e azioni. L’articolo era decisamente a favore della soluzione dichiarativa, sostenendo che dire a un agente quello che deve conoscere è un modo elegante di costruire software. Allen Newell, nell’articolo “The Knowledge Level” (1982), asserisce che gli agenti razionali possono essere descritti e analizzati a un livello astratto definito dalla conoscenza che possiedono piuttosto che dai programmi che eseguono.

La logica stessa ha avuto origine nella filosofia e matematica dell’antica Grecia. Platone discusse la struttura sintattica delle frasi, la loro verità o falsità, il loro significato e la validità delle argomentazioni logiche. Il primo studio sistematico della logica noto è

l’*Organon* di Aristotele, i cui **sillogismi** sono ciò che oggi chiamiamo regole di inferenza, anche se mancavano delle proprietà di composizionalità delle regole usate oggi.

La scuola di Megara e quella stoica iniziarono lo studio sistematico dei connettivi logici di base nel V secolo a.C. Le tavole di verità si devono a Filone di Megara. Gli stoici indicarono cinque regole di inferenza base “valide senza dimostrazione”, e una di esse era quella che oggi chiamiamo Modus Ponens. A partire da quelle cinque hanno derivato un buon numero di altre regole, applicando tra l’altro il teorema di deduzione (Paragrafo 7.5), e definendo il concetto di dimostrazione in modo molto più chiaro di Aristotele. Un buon resoconto della storia della logica delle scuole di Megara e stoica è fornito da Benson Mates (1953).

L'idea di ridurre l'inferenza logica a un processo puramente meccanico applicato a un linguaggio formale risale a Wilhelm Leibniz (1646–1716). George Boole (1847) introdusse il primo sistema completo e funzionale di logica formale nel suo libro *The Mathematical Analysis of Logic*. La logica di Boole era modellata sulla classica algebra dei numeri reali e usava come metodo principale di inferenza la sostituzione di espressioni logicamente equivalenti; non riusciva ancora a esprimere tutta la potenza del calcolo proposizionale, ma altri matematici riempirono presto le lacune restanti. La forma normale congiuntiva fu descritta da Schröder (1877), mentre quella di Horn fu introdotta molto più tardi da Alfred Horn (1951). La prima esposizione completa della logica proposizionale moderna (e di quella del primo ordine) si trova nel *Begriffschrift* di Gottlob Frege (1879).

Il primo dispositivo meccanico capace di svolgere inferenze logiche fu il dimostratore di Stanhope, costruito dal terzo conte di Stanhope (1753–1816). William Stanley Jevons, uno dei matematici che hanno esteso il lavoro di Boole, costruì il suo “pianoforte logico” nel 1869 per eseguire inferenze in logica booleana. Un resoconto divertente e istruttivo di questi primi congegni meccanici di inferenza è fornito da Martin Gardner (1968). I primi programmi per computer dedicati all'inferenza logica sono stati quello di Martin Davis (1954) per le dimostrazioni nell'aritmetica di Presburger (Davis, 1957) e LOGIC THEORIST di Newell, Shaw e Simon (1957).

Emil Post (1921) e Ludwig Wittgenstein (1922) utilizzarono in modo indipendente le tavole di verità come metodo per verificare la validità di formule di logica proposizionale. L'algoritmo Davis-Putnam (Davis e Putnam, 1960) è stato il primo algoritmo per la risoluzione proposizionale, e l'algoritmo DPLL con backtracking (Davis *et al.*, 1962) si rivelò più efficiente. La regola di risoluzione e una dimostrazione della sua completezza furono sviluppate in generalità completa per logica del primo ordine da J. A. Robinson (1965).

Stephen Cook (1971) dimostrò che determinare la soddisfacibilità di una formula nella logica proposizionale (il problema SAT) è un problema NP-completo. Sono noti molti sottoinsiemi della logica proposizionale per cui il problema della soddisfacibilità è risolvibile in tempo polinomiale; le clausole di Horn sono uno di tali sottoinsiemi.

Le prime investigazioni teoriche hanno mostrato che DPLL ha una complessità del caso medio polinomiale per certe distribuzioni comuni di problemi. Ancora meglio, Franco e Paull (1983) hanno mostrato che

gli stessi problemi possono essere risolti in tempo *costante* semplicemente provando assegnamenti casuali. Motivati dal successo empirico della ricerca locale, Koutsoupias e Papadimitriou (1992) hanno mostrato che un semplice algoritmo hill climbing può risolvere quasi tutte le istanze di problemi di soddisfacibilità molto velocemente, suggerendo che i problemi difficili siano rari. Schöning (1999) ha presentato un algoritmo hill climbing randomizzato il cui tempo di esecuzione su problemi 3-SAT è $O(1,333^n)$ nel caso pessimo: ancora esponenziale, ma molto più veloce degli algoritmi precedenti. Il record attuale è $O(1,32216^n)$ (Rolff, 2006).

Sono stati ottenuti rapidi miglioramenti di efficienza nei risolutori proposizionali. Dati dieci minuti di tempo di elaborazione, l'algoritmo DPLL originale nel 1962 riusciva a risolvere problemi con non più di 10 o 15 variabili (su un computer portatile del 2019 riuscirebbe a risolvere problemi con 30 variabili). Nel 1995 il risolutore SATZ (Li e Anbulagan, 1997) riusciva a gestire 1.000 variabili, grazie a strutture dati ottimizzate per indicizzarle. Due contributi fondamentali sono stati la tecnica di indicizzazione di **letterali watched** di Zhang e Stickel (1996), che rende molto efficiente la propagazione delle unità, e l'introduzione di tecniche di apprendimento di clausole (cioè vincoli) dalla comunità CSP di Bayardo e Schrag (1997). Utilizzando queste idee, e stimolati dalla prospettiva di risolvere problemi di verifica di circuiti su scala industriale, Moskewicz *et al.* (2001) svilupparono il risolutore CHAFF, che era in grado di affrontare problemi con milioni di variabili. A partire dal 2002 si sono tenute annualmente gare relative alla risoluzione di problemi SAT; i risolutori vincenti sono nella maggior parte dei casi varianti di CHAFF. L'attuale panorama dei risolutori è illustrato da Gomes *et al.* (2008).

Per tutti gli anni 1980 vari autori provarono ad applicare la ricerca locale al problema della soddisfacibilità, basandosi sull'idea di minimizzare il numero di clausole insoddisfatte (Hansen e Jaumard, 1990). Un algoritmo particolarmente efficace fu sviluppato da Gu (1989) e indipendentemente da Selman *et al.* (1992), che lo chiamò GSAT e dimostrò che era in grado di risolvere una vasta gamma di problemi molto difficili con grande velocità. L'algoritmo WALKSAT descritto in questo capitolo è stato presentato in Selman *et al.* (1996).

La “transizione di fase” nella soddisfacibilità dei problemi k -SAT casuali fu osservata per la prima volta da Simon e Dubois (1989) e ha dato origine a numerosi studi teorici ed empirici, in parte per la connessione con

i fenomeni di transizione di fase nella fisica statistica. Crawford e Auton (1993) localizzarono la transizione 3-SAT a un rapporto clausole/variabili di circa 4,26, osservando che questo coincideva con un picco molto stretto nel tempo di esecuzione del loro risolutore SAT. Cook e Mitchell (1997) offrono un'eccellente rassegna della prima letteratura dedicata al problema.

Algoritmi come **survey propagation** (Parisi e Zecchina, 2002; Maneva *et al.*, 2007) sfruttano speciali proprietà di istanze SAT casuali vicine alla soglia di soddisfacibilità per ottenere prestazioni nettamente migliori dei risolutori SAT generali su tali istanze. Lo stato attuale della conoscenza teorica è illustrato da Achlioptas (2009).

Tra le fonti migliori per informazioni sulla soddisfacibilità, teorica e pratica, vi è l'*Handbook of Satisfiability* (Biere *et al.*, 2009), il fascicolo sulla soddisfacibilità di Donald Knuth (2015) e le periodiche *International Conference on Theory and Applications of Satisfiability Testing*, note come SAT.

L'idea di costruire agenti con la logica proposizionale può essere ricondotta al fondamentale articolo di McCulloch e Pitts (1943), che è ben noto per aver dato origine al campo delle reti neurali, ma in realtà si occupava dell'implementazione di un agente basato su circuiti booleani nel cervello. Stan Rosenschein (Rosenschein, 1985; Kaelbling e Rosenschein, 1990) sviluppò dei modi per compilare agenti basati su circuiti partendo da descrizioni dichiarative dell'ambiente. Rod Brooks (1986, 1989) illustra l'efficacia dei progetti basati su circuiti per controllare i robot (Capitolo 26 del Volume 2). Brooks (1991) sostiene che i progetti basati su circuiti sono tutto ciò che serve per l'IA e che rappresentazione e ragionamento sono pesanti, costosi e non necessari. Nella nostra opinione sono necessari sia il ragionamento, sia i circuiti. Williams *et al.* (2003) descrive un agente ibrido – non troppo diverso dal nostro agente del mondo del wumpus – che controlla veicoli spaziali della NASA, pianificando sequenze di azioni ed effettuando diagnosi e recupero da situazioni di errori.

Il problema generale di tenere traccia di un ambiente parzialmente osservabile è stato presentato nel Capitolo 4 per quanto riguarda le rappresentazioni basate sugli stati. La sua istanziazione per rappresentazioni proposizionali è stata studiata da Amir e Russell (2003), che hanno individuato diverse classi di ambienti che ammettono algoritmi efficienti di stima dello stato e hanno mostrato che, per diverse altre

classi, il problema è intrattabile. Il problema della **proiezione temporale**, che richiede di determinare quali proposizioni rimangono vere dopo che è stata eseguita una sequenza di azioni, può essere visto come un caso particolare della stima dello stato con percezioni vuote. Molti autori hanno studiato questo problema, per la sua importanza nella pianificazione: alcuni risultati relativi alla sua difficoltà sono stati ottenuti da Liberatore (1997). L'idea di rappresentare uno stato-credenza con proposizioni può essere ricondotta a Wittgenstein (1922).

L'approccio alla stima dello stato basata sulla logica usando indici temporali su variabili proposizionali è stato proposto da Kautz e Selman (1992). Generazioni successive di SATPLAN sono state in grado di sfruttare i progressi compiuti nei risolutori SAT e rimangono tra i modi più efficaci di risolvere problemi di pianificazione difficili (Kautz, 2006).

Il **problema del frame** fu individuato per la prima volta da McCarthy e Hayes (1969). Molti studiosi lo consideravano irrisolvibile nella logica del primo ordine, e il problema fece da stimolo per ampie ricerche nelle logiche non monotone. Vari filosofi da Dreyfus (1972) a Crockett (1994) hanno citato il problema del frame come un sintomo dell'inevitabile fallimento dell'intera IA. La soluzione di questo problema con assiomi di stato successore si deve a Ray Reiter (1991). Thielscher (1999) individua il problema inferenziale del frame come concetto separato e fornisce una soluzione. In retrospettiva si può vedere che gli agenti di Rosenschein (1985) utilizzavano circuiti che implementavano assiomi di stato successore, ma Rosenschein non si accorse che con essi il problema del frame poteva darsi in gran parte risolto.

I moderni risolutori proposizionali sono stati applicati a un'ampia varietà di settori industriali, come la sintesi di hardware per computer (Nowick *et al.*, 1993). Il verificatore di soddisfacibilità SATMC è stato usato per rilevare una vulnerabilità precedentemente sconosciuta in un protocollo per l'accesso dell'utente di un browser web (Armando *et al.*, 2008).

Il mondo del wumpus fu inventato da Gregory Yob (1975). Per colmo di ironia, Yob lo sviluppò perché era annoiato dai giochi che utilizzavano una griglia rettangolare: posizionò il suo wumpus su un dodecaedro, mentre noi l'abbiamo riportato nella vecchia e noiosa griglia. Michael Genesereth suggerì che il mondo del wumpus fosse utilizzato come campo di prova per gli agenti.

Logica del primo ordine

- 8.1 Ancora sulla rappresentazione
- 8.2 Sintassi e semantica della logica del primo ordine
- 8.3 Usare la logica del primo ordine
- 8.4 Ingegneria della conoscenza nella logica del primo ordine
- 8.5 Riepilogo
Note storiche e bibliografiche

In cui notiamo che al mondo esiste una grande varietà di oggetti, alcuni dei quali hanno relazioni con altri oggetti, e ci sforziamo di ragionare su di essi.

La logica proposizionale è stata sufficiente per illustrare i concetti base di logica, inferenza e agenti basati sulla conoscenza, ma sfortunatamente è limitata nelle capacità di rappresentazione. In questo capitolo prenderemo in esame la **logica del primo ordine**,¹ che è in grado di rappresentare molto di più e in modo conciso. Cominceremo nel Paragrafo 8.1 con una discussione dei linguaggi di rappresentazione in generale; il Paragrafo 8.2 presenta la sintassi e la semantica della logica del primo ordine; i Paragrafi 8.3 e 8.4 ne illustrano l'uso applicato a semplici rappresentazioni.

8.1 Ancora sulla rappresentazione

In questo paragrafo trattiamo la natura dei linguaggi di rappresentazione. I linguaggi di programmazione (come C++, Java o Python) sono la più grande categoria di linguaggi formali di uso comune. Le strutture dati all'interno dei programmi possono essere usate per rappresentare fatti; per esempio, un programma potrebbe usare una matrice 4×4 per memorizzare il contenuto di un mondo del wumpus. Di conseguenza, un'istruzione nel linguaggio di programmazione come *Mondo[2,2] ← Pozzo* è un modo abbastanza naturale di asserire che c'è un pozzo nella stanza [2,2]. Comporre una stringa con queste istruzioni è sufficiente per eseguire una simulazione del mondo del wumpus.

¹ La logica del primo ordine è chiamata anche **calcolo dei predicati del primo ordine**, e talvolta indicata con l'acronimo inglese **FOL** (*first-order logic*) o **FOPC** (*first-order predicate calculus*).

Ciò che manca ai linguaggi di programmazione è un meccanismo generale che consenta di derivare fatti da altri fatti; ogni aggiornamento a una struttura dati viene effettuato da una procedura specifica del dominio i cui dettagli sono derivati, da parte del programmatore, proprio dalla conoscenza del dominio stesso. Questo approccio procedurale si pone in contrasto con la natura **dichiarativa** della logica proposizionale, che separa nettamente la conoscenza dall'inferenza, e in cui quest'ultima rimane totalmente indipendente dal dominio. I database SQL utilizzano una combinazione di conoscenza dichiarativa e procedurale.

Una seconda lacuna delle strutture dati definite dai programmi (e dai database, peraltro) è l'impossibilità di esprimere facilmente concetti come, per esempio, “c'è un pozzo in una delle stanze [2,2] o [3,1]” oppure “se il wumpus è in [1,1] allora non è in [2,2]”. I programmi possono memorizzare un solo valore per ogni variabile, e alcuni sistemi permettono anche l'uso del valore “indefinito”, ma non possiedono affatto la potenza espressiva necessaria a gestire direttamente informazione parziale.

La logica proposizionale è un linguaggio dichiarativo perché la sua semantica è basata su una relazione di verità che collega le formule e i mondi possibili. Inoltre l'uso della disgiunzione e della negazione lo rende sufficientemente espressivo da gestire informazione parziale. La logica proposizionale ha poi una terza caratteristica desiderabile nei linguaggi di rappresentazione, la **composizionalità**. In un linguaggio composito, il significato di una formula è una funzione del significato delle sue parti. Per esempio, il significato di “ $S_{1,4} \wedge S_{1,2}$ ” è correlato al significato di “ $S_{1,4}$ ” e di “ $S_{1,2}$ ”. Sarebbe ben strano se “ $S_{1,4}$ ” significasse che si sente puzzava nella stanza [1,4] e “ $S_{1,2}$ ” fosse vero se si sente puzzava nella stanza [1,2], ma contemporaneamente “ $S_{1,4} \wedge S_{1,2}$ ” indicasse che la Francia e la Polonia hanno pareggiato 1-1 nella partita di qualificazione di hockey su ghiaccio della settimana scorsa.

Tuttavia, la logica proposizionale in quanto rappresentazione fattorizzata non ha la potenza espressiva per descrivere un ambiente con molti oggetti *in modo conciso*: per esempio, per gestire la presenza di spostamenti d'aria e pozzi siamo stati costretti a scrivere per ogni stanza una regola separata come:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}).$$

D'altra parte, in linguaggio naturale sembra abbastanza facile dire, una volta per tutte: “Nelle stanze adiacenti a quelle che contengono un pozzo si percepisce uno spostamento d'aria”. La sintassi e la semantica dei linguaggi usati dagli esseri umani consentono di descrivere l'ambiente in modo conciso: l'italiano, come la logica del primo ordine, è una rappresentazione strutturata.

8.1.1 Il linguaggio del pensiero

I linguaggi naturali come l'italiano o l'inglese sono effettivamente molto espressivi. Siamo riusciti a scrivere quasi tutto questo libro in un linguaggio naturale, ricorrendo solo occasionalmente ad altri linguaggi (principalmente la matematica e il linguaggio grafico dei diagrammi). Una lunga tradizione di studi nei campi della linguistica e della filosofia considera quello naturale un linguaggio di rappresentazione della conoscenza di tipo dichiarativo. Se fossimo in grado di scoprire le regole del linguaggio naturale, potremmo utilizzarle in sistemi di rappresentazione e ragionamento e trarre vantaggio dai miliardi di pagine che sono state scritte in linguaggio naturale.

Nella visione moderna il linguaggio naturale serve più come mezzo di *comunicazione* che di pura rappresentazione. Quando una persona punta il dito e dice: “Guarda!”, l'ascoltatore viene a sapere che, poniamo, Superman si staglia nel cielo contro il profilo dei grattacieli. Eppure non diremmo che la frase “Guarda!” rappresenta tale fatto. Il significato della frase dipende dal suo contenuto ma anche dal *contesto* in cui è stata pronunciata: chiaramente nessuno penserebbe di inserire una frase come “Guarda!” in una base di conoscenza e di essere in grado di recuperarne il significato senza memorizzare anche una rappresentazione

del contesto; considerazione che solleva la questione di come sia possibile rappresentare il contesto stesso. Infine, i linguaggi naturali hanno anche il difetto di contenere *ambiguità*, un problema per un linguaggio di rappresentazione. Come dice Pinker (1995): “Quando le persone pensano alla parola *spring*, certamente non devono chiedersi se stanno pensando a una stagione (*spring* come ‘primavera’) o a un oggetto che fa *boing* (*spring* come ‘molla’); e se a una parola possono corrispondere due pensieri, i pensieri non possono essere parole”.

La famosa **ipotesi di Sapir-Whorf** (Whorf, 1956) afferma che la nostra comprensione del mondo è fortemente influenzata dal linguaggio che parliamo. È certamente vero che comunità linguistiche diverse suddividono il mondo in maniera diversa. In francese ci sono due parole, “chaise” e “fauteuil”, per descrivere un concetto che in inglese si esprime con una parola sola: “chair”. Tuttavia chi parla inglese è in grado di riconoscere facilmente la categoria *fauteuil* e di assegnarle un nome, qualcosa come “open-arm chair”, perciò il linguaggio fa realmente differenza? Whorf si basava principalmente su intuizione e speculazione, e le sue idee sono state per lo più abbandonate, ma negli anni a venire abbiamo avuto a disposizione dati reali ricavati da studi antropologici, psicologici e neurologici.

Per esempio, ricordate quale delle due frasi seguenti costituisce l'apertura del Paragrafo 8.1?

“In questo paragrafo trattiamo la natura dei linguaggi di rappresentazione. . .”

“Questo paragrafo tratta i linguaggi di rappresentazione della conoscenza. . .”

Wanner (1974) fece un esperimento simile e scoprì che le persone facevano la scelta giusta in una percentuale praticamente casuale (circa il 50% dei casi), ma ricordavano il contenuto di ciò che avevano letto con una precisione superiore al 90%. Questo suggerisce che le persone interpretano le parole che leggono e formano una rappresentazione *non verbale*, e che le parole esatte non sono rilevanti.

Ancora più interessante è il caso in cui un concetto è del tutto assente da un linguaggio. Chi parla il linguaggio nativo degli aborigeni australiani, Guugu Yimithirr, non dispone di parole per indicazioni di direzione relative (o *egocentriche*) quali davanti, dietro, destra o sinistra, ma utilizza indicazioni di direzione assolute, dicendo, per esempio, l'equivalente di: “Mi fa male il braccio a nord”. Questa differenza nei linguaggi porta a una differenza nel comportamento: chi parla il Guugu Yimithirr è più bravo a orientarsi in campo aperto, mentre chi parla italiano è più bravo a mettere la forchetta a sinistra del piatto.

Il linguaggio sembra anche influenzare il pensiero attraverso caratteristiche grammaticali apparentemente arbitrarie quali il genere dei nomi. Per esempio, “ponte” è maschile in spagnolo e femminile in tedesco. Boroditsky (2003) chiese a una serie di persone di scegliere aggettivi inglesi per descrivere una fotografia di un particolare ponte. Le persone di lingua spagnola scelsero *big*, *dangerous*, *strong* e *towering*, mentre quelle di lingua tedesca scelsero *beautiful*, *elegant*, *fragile* e *slender*.

Le parole possono servire quali punti di ancoraggio che influenzano il modo in cui percepiamo il mondo. Loftus e Palmer (1974) mostraroni ai partecipanti a un esperimento un film di un incidente automobilistico. Le persone a cui fu chiesto: “A che velocità viaggiavano le automobili quando si toccarono tra loro?” risposero con una media di 32 mph, mentre quelle a cui fu posta la domanda utilizzando la parola “scontrarono” invece di “toccarono” risposero con una media di 41 mph per le stesse automobili e lo stesso filmato. In generale, esistono differenze misurabili ma piccole nell'elaborazione cognitiva svolta da coloro che parlano lingue diverse, ma non vi sono evidenze convincenti del fatto che ciò porti a una differenza notevole nella visione del mondo.

In un sistema di ragionamento logico che utilizza la forma normale congiuntiva (CNF), possiamo vedere che le forme linguistiche “ $\neg(A \vee B)$ ” e “ $\neg A \wedge \neg B$ ” coincidono perché possiamo osservare all'interno del sistema e vedere che le due formule sono memorizzate con la stessa forma canonica CNF. E si inizia a vedere la possibilità di fare qualcosa di simile con il ragionamento umano. Mitchell *et al.* (2008) hanno fatto un esperimento sottponendo

alcune persone a una risonanza magnetica funzionale e mostrando loro parole come “sedano” mentre la macchina forniva immagini del loro cervello. Un programma di apprendimento automatico addestrato su coppie (parola, immagine) è stato in grado di predire correttamente nel 77% dei casi i risultati di attività a scelta binaria (per esempio scegliere tra “sedano” o “aeroplano”). Il sistema è anche in grado di fare previsioni con risultati superiori rispetto a previsioni casuali per parole per le quali non ha mai visto prima un’immagine di risonanza magnetica funzionale (considerando le immagini corrispondenti a parole correlate) e per persone che non ha mai visto prima (il che dimostra che la risonanza magnetica funzionale rivela un livello di rappresentazione comune a tutte le persone). Questo tipo di lavoro è ancora in una fase iniziale, ma la risonanza magnetica funzionale – e altre tecnologie di *imaging* come l’elettrofisiologia intracranica (Sahin *et al.*, 2009) – promette di fornirci idee molto più concrete riguardo le rappresentazioni della conoscenza umana.

Dal punto di vista della logica formale, rappresentare la stessa conoscenza in due modi diversi non fa alcuna differenza; gli stessi fatti saranno ricavabili da ognuna delle due rappresentazioni. Nella pratica, tuttavia, una rappresentazione potrebbe richiedere meno passi per ricavare una conclusione, perciò un “ragionatore” con risorse limitate potrebbe arrivare alla conclusione utilizzando una rappresentazione ma non utilizzando l’altra. Per compiti *non deduttivi* quali l’apprendimento dall’esperienza, i risultati dipendono *necessariamente* dalla forma delle rappresentazioni usate. Vedremo nel Capitolo 19 del Volume 2 che, quando un programma di apprendimento considera due teorie possibili del mondo, entrambe consistenti con tutti i dati, il modo più comune per sceglierne una è quello di scegliere la teoria più succinta, che dipende dal linguaggio utilizzato per rappresentare le teorie. Perciò, l’influenza del linguaggio sul pensiero è inevitabile per qualsiasi agente che utilizza l’apprendimento.

8.1.2 Mettere insieme il meglio dei linguaggi formali e naturali

Possiamo partire da un aspetto fondamentale della logica proposizionale – una semantica dichiarativa e compositiva, indipendente dal contesto e non ambigua – e su tali fondamenti costruire una logica più espressiva, prendendo in prestito metodi di rappresentazione dal linguaggio naturale ed evitando le sue limitazioni. Quando prendiamo in esame la sintassi di un linguaggio naturale, gli elementi che si distinguono più facilmente sono i sostantivi e i sintagmi nominali che si riferiscono a **oggetti** (stanze, pozzi, wumpus) e i verbi e i sintagmi verbali, con aggettivi e avverbi, che si riferiscono a **relazioni** tra gli oggetti (è ventosa, è adiacente a, scocca). Alcune relazioni, dette **funzioni**, hanno un solo “valore” per ogni “input”. È facile elencare alcuni esempi di oggetti, relazioni e funzioni.

oggetto
relazione
funzione

proprietà

- Oggetti: persone, cavalli, numeri, teorie, Ronald McDonald, colori, partite a baseball, guerre, secoli . . .
- Relazioni: possono essere unarie, nel qual caso prendono il nome di **proprietà** (come quella di essere rossi, tondi, falsi, primi, alti). Le più generali sono *n*-arie e comprendono relazioni come fratello di, più grande di, all’interno di, parte di, avvenuto dopo, di colore, possiede, sta in mezzo a . . .
- Funzioni: padre di, miglior amico, terzo inning di, uno più di, all’inizio di . . .

In effetti, quasi tutte le asserzioni possono essere ricondotte agli oggetti e alle loro proprietà e relazioni. Vediamo alcuni esempi.

- “Uno più due fa tre”.

Oggetti: uno, due, tre, uno più due. Relazione: fa. Funzione: più (“uno più due” è il nome dell’oggetto che si ottiene applicando la funzione “più” agli oggetti “uno” e “due”, “tre” è un altro nome dello stesso oggetto).

- “Le stanze adiacenti a quella che contiene un wumpus sono puzzolenti”.
Oggetti: wumpus, stanze. Proprietà: puzzolente. Relazione: adiacenti.
- “Il malvagio Re Giovanni governò l’Inghilterra durante il 1200”.
Oggetti: Giovanni, Inghilterra, 1200. Relazione: governò durante. Proprietà: malvagio, re.

Il linguaggio della **logica del primo ordine**, di cui definiremo sintassi e semantica nel prossimo paragrafo, è costruito intorno agli oggetti e alle relazioni. La sua importanza nei campi della matematica, della filosofia e dell’intelligenza artificiale deriva proprio dal fatto che tali discipline (e, in effetti, gran parte dell’esistenza umana di tutti i giorni) possono essere proficuamente pensate come attività che hanno a che fare con oggetti e con le relazioni che li collegano. La logica del primo ordine può anche esprimere fatti che riguardano *alcuni* o *tutti* gli oggetti dell’universo. Questo rende possibile esprimere leggi generali o regole, come “le stanze adiacenti a un wumpus puzzano”.

La differenza fondamentale tra la logica proposizionale e quella del primo ordine sta nell’**impegno ontologico** assunto da ognuno dei due linguaggi, ovvero le sue ipotesi circa la natura della *realità*. Matematicamente, questo impegno è espresso attraverso la natura dei **modelli** formali rispetto ai quali è definita la verità delle formule. La logica proposizionale, per esempio, dà per scontato che i fatti nel mondo sono veri oppure no. Ogni fatto può essere in uno di due possibili stati, vero o falso, e ogni modello assegna *true* o *false* a ogni simbolo proposizionale (cfr. Paragrafo 7.4.2). La logica del primo ordine fa ulteriori ipotesi, e in particolare che il mondo consista di oggetti legati da relazioni che possono o meno essere verificate (cfr. Figura 8.1). I modelli formali sono di conseguenza più complessi di quelli della logica proposizionale.

impegno ontologico

L’impegno ontologico è un importante punto di forza della logica (proposizionale e del primo ordine) perché ci consente di partire da affermazioni vere e inferire altre affermazioni vere. È particolarmente potente in domini in cui ogni proposizione ha confini ben chiari, come la matematica o il mondo del wumpus in cui una stanza contiene un pozzo oppure non lo contiene, non sono possibili stanze con una rientranza vagamente somigliante a un pozzo. Nel mondo reale, però, molte proposizioni hanno confini vaghi: Vienna è una grande città? Questo ristorante serve piatti prelibati? Quella persona è alta? Le risposte dipendono dai destinatari delle domande, e a volte possono essere non dirimenti.

Una possibilità è quella di raffinare la rappresentazione: se una linea di separazione netta che divide le città in “grandi” e “non grandi” esclude troppe informazioni per l’applicazione che ci interessa, è possibile aumentare il numero di categorie dimensionali, oppure usare un simbolo di funzione *Popolazione*. Un’altra soluzione è fornita dalla **logica sfumata** o **fuzzy**, in cui vi è l’impegno ontologico che le proposizioni hanno un **grado di verità** compreso tra 0 e 1. Per esempio, la frase “Vienna è una grande città” nel nostro mondo potrebbe avere un grado di verità di 0,8 in logica fuzzy, mentre “Parigi è una grande città” potrebbe avere un grado di verità di 0,9. Questo meccanismo corrisponde meglio al modo in cui concepiamo

logica fuzzy
grado di verità

linguaggio	impegno ontologico (ciò che esiste nel mondo)	impegno epistemologico (le credenze di un agente circa un fatto)
logica proposizionale	fatti	vero/falso/sconosciuto
logica del primo ordine	fatti, oggetti, relazioni	vero/falso/sconosciuto
logica temporale	fatti, oggetti, relazioni, tempi	vero/falso/sconosciuto
teoria della probabilità	fatti	grado di credenza $\in [0, 1]$
logica fuzzy	fatti con gradi di verità $\in [0, 1]$	intervallo di valori conosciuto

Figura 8.1 Linguaggi formali con i rispettivi impegni ontologici ed epistemologici.

logica temporale**logica di ordine superiore****impegni epistemologici****dominio
elementi
del dominio**

intuitivamente il mondo, ma rende più difficile l'inferenza: invece di una sola regola per determinare il valore di verità di $A \mid B$, la logica fuzzy richiede diverse regole a seconda del dominio. Un'altra possibilità, che tratteremo nel Paragrafo 24.1 (Volume 2), è quella di assegnare ogni concetto a un punto in uno spazio multidimensionale e poi misurare la distanza tra il concetto “grande città” e il concetto “Vienna” o il concetto “Parigi”.

Varie logiche specializzate si impegnano ontologicamente ancora di più; la **logica temporale**, per esempio, parte dal presupposto che i fatti siano verificati in un determinato *momento* e che i tempi (che possono essere istanti o intervalli) siano tutti ordinati. In questo modo le logiche specializzate possono conferire a certe categorie di oggetti (e agli assiomi che li riguardano) uno stato “privilegiato”, invece di definirli semplicemente nella base di conoscenza. Le **logiche di ordine superiore** considerano oggetti le stesse relazioni e funzioni a cui la logica di primo ordine fa riferimento. In questo modo è possibile formulare enunciati che riguardano *tutte* le relazioni, definendo per esempio che cosa si intende per transitività di una relazione. A differenza della maggior parte delle logiche specializzate, quelle di ordine superiore sono più espressive della logica di primo ordine in senso stretto, il che significa che alcune loro formule non possono essere espresse con un numero finito di formule della logica del primo ordine.

Una logica può anche essere caratterizzata dai suoi **impegni epistemologici**, ovvero dai diversi stati di conoscenza che permette nei confronti di ciascun fatto. Sia nella logica proposizionale che in quella del primo ordine, una formula rappresenta un fatto che l'agente può ritenere vero, oppure falso; inoltre può anche non avere un'opinione. In queste logiche, quindi, a ogni formula può essere associato uno di tre possibili stati di conoscenza.

I sistemi che usano la **teoria della probabilità**, d'altra parte, permettono qualsiasi *grado di credenza*, da 0 (totalmente non creduto) a 1 (totalmente creduto). È importante evitare di confondere il grado di credenza della teoria della probabilità con il grado di verità della logica fuzzy. In effetti, alcuni sistemi fuzzy permettono di esprimere incertezza (grado di credenza) sui gradi di verità. Per esempio, un agente in un mondo del wumpus probabilistico potrebbe credere che il mostro si trovi in [1,3] con probabilità 0,75 e in [2,3] con probabilità 0,25 (anche se il wumpus si trova certamente in una particolare stanza).

8.2 Sintassi e semantica della logica del primo ordine

Cominciamo specificando meglio il modo in cui i mondi possibili della logica del primo ordine riflettono l'impegno ontologico nei confronti di oggetti e relazioni. Introdurremo quindi i vari elementi del linguaggio, spiegando man mano la loro semantica. I punti principali sono relativi alle modalità con cui in cui il linguaggio facilita rappresentazioni concise e conduce a procedure di ragionamento corrette.

8.2.1 Modelli per la logica del primo ordine

Nel Capitolo 7 si è detto che i modelli di un linguaggio logico sono le strutture formali che costituiscono i mondi possibili che possono essere presi in considerazione. Ogni modello collega il vocabolario delle formule logiche a elementi del mondo possibile, in modo che possa essere determinata la verità di qualsiasi formula. Quindi, i modelli della logica proposizionale collegano simboli a valori di verità predefiniti. I modelli della logica del primo ordine sono molto più interessanti: per prima cosa, contengono oggetti! Il **dominio** di un modello è l'insieme degli oggetti che contiene, chiamati appunto **elementi del dominio**. Il dominio *non deve essere vuoto*: ogni mondo possibile deve contenere almeno un oggetto (cfr. l'Esercizio 8.EMPT per una discussione sui mondi vuoti). In termini matematici, non importa *che cosa* sono questi oggetti – tutto ciò che conta è *quanti* ve ne sono in ogni particolare

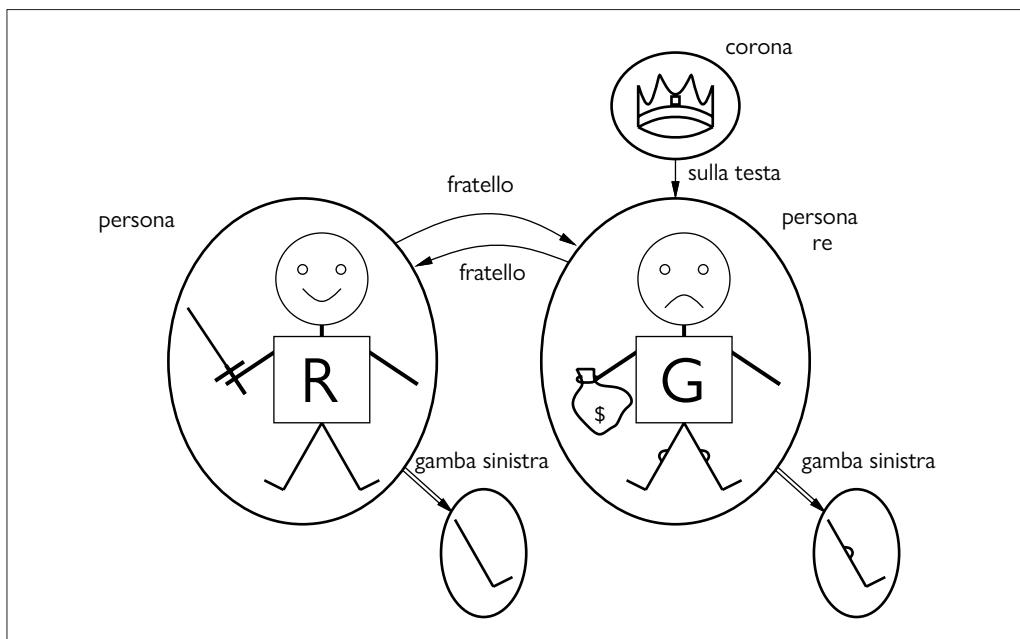


Figura 8.2
Un modello che contiene cinque oggetti, due relazioni binarie (fratello e sulla testa), tre relazioni unarie (persona, re e corona) e una funzione unaria (gamba sinistra).

modello – ma per scopi pedagogici utilizzeremo un esempio concreto. La Figura 8.2 mostra un modello con cinque oggetti: Riccardo Cuor di Leone, Re d'Inghilterra dal 1189 al 1199; suo fratello minore, il malvagio Re Giovanni, che regnò dal 1199 al 1215; le gambe sinistre di Riccardo e Giovanni; e una corona.

Gli oggetti del modello possono essere messi in relazione in molti modi. Nella figura, Riccardo e Giovanni sono fratelli. Formalmente, una relazione non è altro che un insieme di **tupla** di oggetti collegati (una tupla è una collezione ordinata di oggetti, e si scrive racchiusa tra parentesi angolari). Così, la relazione di “fratellanza” nel modello è costituita dall’insieme:

$$\{ \langle \text{Riccardo Cuor di Leone}, \text{Re Giovanni} \rangle, \langle \text{Re Giovanni}, \text{Riccardo Cuor di Leone} \rangle \}. \quad (8.1)$$

Notate che qui abbiamo indicato i nomi per esteso ma, se preferite, potete sostituire mentalmente le immagini della figura al loro posto. La corona è sulla testa di Giovanni, per cui la relazione “sulla testa” contiene solo una tupla, $\langle \text{la corona}, \text{Re Giovanni} \rangle$. Le relazioni “fratello” e “sulla testa” sono binarie; questo significa che collegano coppie di oggetti. Il modello contiene anche relazioni unarie, o proprietà: “persona” è vera sia per Riccardo che per Giovanni; la proprietà “re” vale solo per Giovanni (presumibilmente perché, a questo punto, Riccardo è già morto); infine, la proprietà “corona” è vera solo per la corona.

Alcuni tipi di relazioni si possono considerare meglio come funzioni, in quanto ogni dato oggetto può essere collegato attraverso di esse a esattamente un altro oggetto. Per esempio, ogni persona ha una gamba sinistra, ragion per cui il modello ha una funzione unaria “gamba sinistra” che collega una tupla di un solo elemento a un oggetto e che include le seguenti corrispondenze:

$$\begin{aligned} &\langle \text{Riccardo Cuor di Leone} \rangle \rightarrow \text{la gamba sinistra di Riccardo} \\ &\langle \text{Re Giovanni} \rangle \rightarrow \text{la gamba sinistra di Giovanni}. \end{aligned} \quad (8.2)$$

Se vogliamo essere formali, dobbiamo dire che i modelli nella logica del primo ordine richiedono **funzioni totali**, ovvero che ci dev’essere un valore per ogni tupla di input. Questo significa che anche la corona deve avere una gamba sinistra, e anche tutte le gambe devono

funzione totale

<i>Formula</i>	\rightarrow	<i>FormulaAtomica</i> <i>FormulaComplessa</i>
<i>FormulaAtomica</i>	\rightarrow	<i>Predicato</i> <i>Predicato(Termine, ...)</i> <i>Termine = Termine</i>
<i>FormulaComplessa</i>	\rightarrow	(<i>Formula</i>)
		\neg <i>Formula</i>
		<i>Formula</i> \wedge <i>Formula</i>
		<i>Formula</i> \vee <i>Formula</i>
		<i>Formula</i> \Rightarrow <i>Formula</i>
		<i>Formula</i> \Leftrightarrow <i>Formula</i>
		<i>Quantificatore Variabile, ... Formula</i>
<i>Termine</i>	\rightarrow	<i>Funzione(Termine, ...)</i>
		<i>Costante</i>
		<i>Variabile</i>
<i>Quantificatore</i>	\rightarrow	\forall \exists
<i>Costante</i>	\rightarrow	<i>A</i> <i>X</i> ₁ <i>Giovanni</i> ...
<i>Variabile</i>	\rightarrow	<i>a</i> <i>x</i> <i>s</i> ...
<i>Predicato</i>	\rightarrow	<i>True</i> <i>False</i> <i>Dopo</i> <i>Ama</i> <i>Piove</i> ...
<i>Funzione</i>	\rightarrow	<i>Madre</i> <i>GambaSinistra</i> ...
PRECEDENZA OPERATORI	:	$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Figura 8.3 La sintassi della logica del primo ordine con uguaglianza, espressa nella forma di Backus–Naur (cfr. Appendice B se non avete familiarità con questa notazione). Sono specificate le precedenze degli operatori, dalla più alta alla più bassa. La precedenza dei quantificatori è tale che un quantificatore prevale su tutto ciò che è alla sua destra.

a loro volta avere una gamba sinistra. Per risolvere questo problema si può usare un artificio che prevede l’aggiunta di un oggetto “invisibile” che fa le veci di gamba sinistra di tutti gli oggetti che non hanno effettivamente una gamba sinistra, incluso l’oggetto stesso. Fortunatamente, finché nessuno esprime formule che riguardano le gambe sinistre degli oggetti che non hanno gambe, questi particolari tecnici sono superflui.

Fin qui abbiamo descritto gli elementi dei modelli per la logica del primo ordine. L’altra parte essenziale di un modello è il collegamento tra questi elementi e il vocabolario delle formule logiche, che spieghiamo nel seguito.

8.2.2 Simboli e interpretazioni

Passiamo ora a considerare la sintassi della logica del primo ordine. Il lettore impaziente può leggere una descrizione completa dalla grammatica formale nella Figura 8.3.

simbolo di costante Gli elementi sintattici base sono i simboli utilizzati per indicare gli oggetti, le relazioni e le funzioni. Di conseguenza ce ne sono tre tipi: **simboli di costante**, che rappresentano oggetti; **simboli di predicato**, che rappresentano relazioni; **simboli di funzione**, che rappresentano appunto funzioni. Adotteremo la convenzione che i simboli comincino sempre con lettere maiuscole: useremo quindi simboli di costante come *Riccardo* e *Giovanni*; simboli di predicato come *Fratello*, *SullaTesta*, *Persona*, *Re* e *Corona*; e il simbolo di funzione *GambaSinistra*. Come nel caso dei simboli proposizionali, la scelta dei nomi è totalmente libera. Ogni simbolo di predicato e di funzione ha una specifica **arità** che indica il numero di argomenti.

simbolo di predicato

simbolo di funzione

arità

Ogni modello deve fornire le informazioni richieste per determinare se una formula data è vera o falsa. Perciò, oltre a oggetti, relazioni e funzioni, ogni modello include un'**interpretazione** che specifica esattamente a quali oggetti, relazioni e funzioni fanno riferimento i simboli di costante, predicato e funzione. Un'interpretazione possibile per il nostro modello – che uno studioso di logica chiamerebbe **interpretazione intesa** – è la seguente:

interpretazione

- *Riccardo* si riferisce a Riccardo Cuor di Leone e *Giovanni* al malvagio Re Giovanni;
- *Fratello* indica la relazione di fratellanza, ovvero l'insieme di tuple di oggetti fornite dall'Equazione (8.1); *SullaTesta* è una relazione tra la corona e Re Giovanni; *Persona*, *Re* e *Corona* fanno riferimento agli insiemi di oggetti che sono effettivamente persone, re e corone;
- *GambaSinistra* indica la funzione “gamba sinistra” come definita dall'Equazione (8.2).

interpretazione
intesa

Naturalmente ci sono molte altre interpretazioni possibili. Per esempio, potremmo chiamare *Riccardo* la corona e *Giovanni* la gamba sinistra di Re Giovanni. Dato che nel modello ci sono cinque oggetti, solo per i simboli di costante *Riccardo* e *Giovanni* le interpretazioni possibili sono 25. Notate che non è obbligatorio che tutti gli oggetti abbiano un nome; la nostra interpretazione intesa, per esempio, non specifica quelli delle gambe e della corona.² È anche possibile che un oggetto abbia più nomi; in una certa interpretazione sia *Riccardo* che *Giovanni* potrebbero riferirsi alla corona. Se vi sembra che questo possa portare a confusione, ricordate che anche nella logica proposizionale è perfettamente possibile che ci siano modelli in cui i simboli *Nuvoloso* e *Soleggiato* sono entrambi veri; è compito della base di conoscenza escludere i modelli inconsistenti.

Per riassumere, un modello nella logica del primo ordine è costituito da un insieme di oggetti e da un'interpretazione che associa simboli di costante a oggetti, simboli di funzione a funzioni su questi oggetti e simboli di predicato a relazioni. Come per la logica proposizionale, conseguenza logica, validità e così via sono definiti in riferimento a *tutti i modelli possibili*. Per farvi un'idea di come appare l'insieme di tutti i modelli possibili, osservate la Figura 8.4, dove si vede che i modelli variano nel numero di oggetti che contengono (da uno fino a infiniti) e nel modo in cui i simboli di costante si associano a oggetti.

Poiché il numero di modelli possibili non ha limiti, non è possibile verificare la conseguenza logica enumerando tutti i modelli possibili (come abbiamo fatto nella logica proposizionale). Anche se il numero degli oggetti è limitato, il numero di combinazioni può essere molto grande (cfr. Esercizio 8.MCNT). Nell'esempio della Figura 8.4 vi sono 137.506.194.466 modelli con al più sei oggetti.

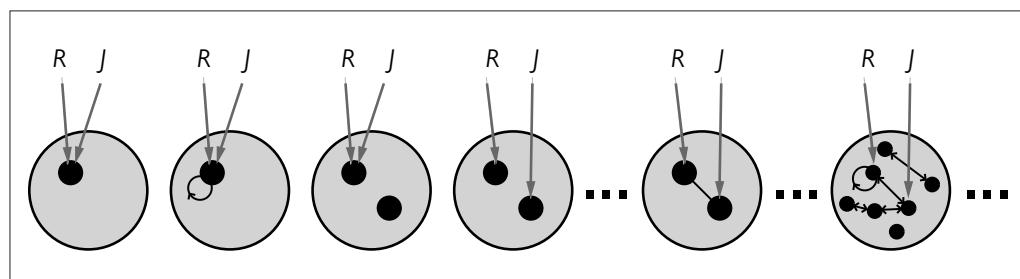


Figura 8.4 Alcuni membri dell'insieme di tutti i modelli per un linguaggio con due simboli di costante *R* e *J* e un solo simbolo di relazione binaria. L'interpretazione di ogni simbolo di costante è mostrata da una freccia in grigio. All'interno di ciascun modello, gli oggetti in relazione fra loro sono connessi da frecce.

² Più avanti, nel Paragrafo 8.2.8, esamineremo una semantica in cui ogni oggetto ha esattamente un nome.

8.2.3 Termini

Un **termine** è un'espressione logica che si riferisce a un oggetto. I simboli di costante sono termini, ma non è sempre il caso di assegnare un simbolo distinto a ogni oggetto: per esempio, nel linguaggio di tutti i giorni useremo l'espressione “la gamba sinistra di Re Giovanni” e non le daremo certo un nome specifico. Questo è proprio lo scopo dei simboli di funzione: invece di usare un simbolo di costante, potremo scrivere *GambaSinistra(Giovanni)*.³

Nel caso generale, un termine complesso è formato da un simbolo di funzione seguito da una lista tra parentesi dei suoi argomenti, costituiti a loro volta da termini. È importante tener sempre presente che un termine complesso è solo un modo complicato di dare il nome a un oggetto: non è affatto una “chiamata di subroutine” che “restituisce un valore”. Non c'è una procedura *GambaSinistra* che prende come input una persona e restituisce una gamba. Possiamo ragionare sulle gambe sinistre (per esempio affermando la regola generale che tutte le persone ne hanno una, e quindi deducendo che anche Giovanni deve averla) senza mai neppure fornire una definizione di *GambaSinistra*. Questa è una cosa decisamente impossibile da fare con le procedure dei linguaggi di programmazione.

La semantica formale dei termini si spiega senza difficoltà. Consideriamo un termine $f(t_1, \dots, t_n)$. Il simbolo di funzione f si riferisce a una qualche funzione del modello (che chiameremo F); i termini usati come argomento fanno riferimento a oggetti del dominio (che indicheremo con d_1, \dots, d_n); nella sua interezza il termine indica quindi l'oggetto che corrisponde al valore della funzione F applicata a d_1, \dots, d_n . Per esempio, supponiamo che il simbolo di funzione *GambaSinistra* si riferisca alla funzione mostrata nell'Equazione (8.2) e che *Giovanni* sia Re Giovanni: allora *GambaSinistra(Giovanni)* sarà la gamba sinistra di Re Giovanni. In questo modo, l'interpretazione fissa l'oggetto corrispondente a ogni termine.

8.2.4 Formule atomiche

formula atomica
atomo

Ora che disponiamo dei termini per riferirci agli oggetti e dei simboli di predicato per riferirci alle relazioni, possiamo mettere insieme i due elementi per dare origine a **formule atomiche** capaci di asserire fatti. Una **formula atomica** (o **atomo**, per brevità) è composta da un simbolo di predicato seguito da una lista di termini tra parentesi:

Fratello(Riccardo, Giovanni).

La formula afferma, in base all'interpretazione intesa che abbiamo fornito, che Riccardo Cuor di leone è un fratello di Re Giovanni.⁴ Le formule atomiche possono avere termini complessi come argomenti. Così,

Sposato(Padre(Riccardo), Madre(Giovanni))

afferma (ancora una volta, secondo un'interpretazione adeguata) che il padre di Riccardo Cuor di Leone è sposato con la madre di Re Giovanni.⁵

Una formula atomica è vera in un dato modello se la relazione a cui fa riferimento il simbolo di predicato è verificata tra gli oggetti a cui fanno riferimento gli argomenti.



³ Le **espressioni** λ forniscono una notazione utile per costruire nuovi simboli di funzione “al momento”. La funzione che restituisce il quadrato del suo parametro, ad esempio, può essere scritta come $(\lambda x. x \times x)$ e può essere applicata agli argomenti proprio come qualsiasi altro simbolo di funzione. Un'espressione λ può anche essere definita e utilizzata come simbolo di predicato. L'operatore `lambda` in Lisp e Python funziona esattamente in questo modo. Notate che l'uso di λ in questo modo *non* aumenta il potere espressivo della logica del primo ordine, dal momento che ogni formula che include un'espressione λ può essere sostituita da una formula equivalente che non ne fa uso.

⁴ Seguiremo la convenzione di ordinamento degli argomenti secondo cui $P(x, y)$ si legge “ x è un P di y ”.

⁵ Questa ontologia riconosce un solo padre e una sola madre per ogni persona. Un'ontologia più complessa potrebbe riconoscere madre biologica, madre naturale, madre adottiva, ecc.

8.2.5 Formule complesse

Proprio come nella logica proposizionale, possiamo usare i **connettivi logici** per costruire formule complesse (o composte). Ecco quattro formule che, secondo la nostra interpretazione intesa, sono vere nel modello della Figura 8.2:

$\neg \text{Fratello}(\text{GambaSinistra}(\text{Riccardo}), \text{Giovanni})$
 $\text{Fratello}(\text{Riccardo}, \text{Giovanni}) \wedge \text{Fratello}(\text{Giovanni}, \text{Riccardo})$
 $\text{Re}(\text{Riccardo}) \vee \text{Re}(\text{Giovanni})$
 $\neg \text{Re}(\text{Riccardo}) \Rightarrow \text{Re}(\text{Giovanni})$.

8.2.6 Quantificatori

Avendo a disposizione una logica basata sugli oggetti, è naturale che si desideri esprimere caratteristiche di intere collezioni di oggetti senza doverli enumerare uno per uno. I **quantificatori** ci permettono di farlo. La logica del primo ordine comprende due quantificatori standard, quello *universale* e quello *esistenziale*.

quantificatore

Quantificazione universale (\forall)

Come ricorderete, nel Capitolo 7 abbiamo avuto molte difficoltà nell'esprimere leggi generali per mezzo della logica proposizionale. Regole come “le stanze adiacenti a quella che contiene il wumpus sono puzzolenti” e “tutti i re sono persone” sono, al contrario, molto comuni nella logica del primo ordine. Ci occuperemo della prima nel Paragrafo 8.3; la seconda si scrive:

$$\forall x \text{ Re}(x) \Rightarrow \text{Persona}(x).$$

Il **quantificatore universale** \forall si legge “per ogni...”; per cui l'intera formula afferma che “per ogni x , se x è un re, allora x è una persona”. Il simbolo x prende il nome di **variabile**. Per convenzione, le variabili si scrivono in minuscolo. Una variabile da sola è anche un termine, e per questo motivo può anche essere usata come argomento di una funzione, come in $\text{GambaSinistra}(x)$. Un termine che non comprende variabili si chiama **termine ground**.

quantificatore
universale
variabile

Intuitivamente la formula $\forall x P$, dove P è una qualsiasi formula logica, afferma che P è vera per ogni oggetto x . Più precisamente, $\forall x P$ è vera in un dato modello se P è vera in tutte le possibili **interpretazioni estese** costruite a partire dall'interpretazione fornita nel modello, dove ogni interpretazione estesa specifica un elemento del dominio a cui x fa riferimento.

termine ground

Tutto ciò sembra molto complicato, ma in effetti è solo un modo formale di definire il significato intuitivo della quantificazione universale. Considerate il modello della Figura 8.2 e la sua interpretazione intesa. Possiamo estendere tale interpretazione in cinque modi:

interpretazione
estesa

- $x \rightarrow \text{Riccardo Cuor di Leone}$
- $x \rightarrow \text{Re Giovanni}$
- $x \rightarrow \text{la gamba sinistra di Riccardo}$
- $x \rightarrow \text{la gamba sinistra di Giovanni}$
- $x \rightarrow \text{la corona.}$

La formula universalmente quantificata $\forall x \text{ Re}(x) \Rightarrow \text{Persona}(x)$ è vera nel modello originale se la formula $\text{Re}(x) \Rightarrow \text{Persona}(x)$ è vera in ognuna delle cinque interpretazioni estese. In altre parole, la formula universalmente quantificata è equivalente all'asserzione delle cinque formule seguenti.

Riccardo Cuor di Leone è un re \Rightarrow Riccardo Cuor di Leone è una persona.

Re Giovanni è un re \Rightarrow Re Giovanni è una persona.

La gamba sinistra di Riccardo è un re \Rightarrow la gamba sinistra di Riccardo è una persona.

La gamba sinistra di Giovanni è un re \Rightarrow la gamba sinistra di Giovanni è una persona.

La corona è un re \Rightarrow la corona è una persona.

Esaminiamo attentamente questo insieme di asserzioni. Dato che, nel nostro modello, l'unico re è Giovanni, la seconda formula afferma che egli è una persona, come ci potremmo aspettare. Ma che dire delle altre quattro formule, che sembrano ragionare di gambe e corone? Anch'esse fanno parte del significato di "tutti i re sono persone"? In effetti, anche le altre quattro asserzioni sono vere nel modello, ma non affermano nulla sulla natura umana di gambe e corone, e neppure dello stesso Riccardo. La ragione sta nel fatto che nessuno di questi oggetti è un re. Guardando la tavola di verità di \Rightarrow (Figura 7.8 del Capitolo 7), possiamo vedere che un'implicazione è sempre vera quando le sue premesse sono false, in modo *del tutto indipendente* dalla verità della conclusione. Così, scrivendo la formula universalmente quantificata, che equivale a un'intera lista di formule singole, asseriamo la verità della conclusione solo per quegli oggetti per cui è vera la premessa, e non diciamo nulla di tutti quelli per cui la premessa è falsa. La definizione di \Rightarrow della tavola di verità risulta quindi ideale per scrivere regole generali mediante l'uso di quantificatore universali.

Un errore comune, commesso frequentemente anche dai lettori diligenti che hanno letto più volte questo paragrafo, è usare la congiunzione al posto dell'implicazione. La formula:

$$\forall x \text{Re}(x) \wedge \text{Persona}(x)$$

sarebbe equivalente ad asserire:

Riccardo Cuor di Leone è un re \wedge Riccardo Cuor di Leone è una persona;

Re Giovanni è un re \wedge Re Giovanni è una persona;

la gamba sinistra di Riccardo è un re \wedge la gamba sinistra di Riccardo è una persona;

e così via. Naturalmente, non è questo ciò che vogliamo affermare.

Quantificazione esistenziale (\exists)

quantificatore esistenziale

La quantificazione universale permette di scrivere formule che riguardano tutti gli oggetti. In modo analogo, è possibile formulare enunciati circa *alcuni* oggetti senza citarli per nome, usando un **quantificatore esistenziale**. Per esempio, per dire che Re Giovanni ha una corona sulla testa, scriviamo:

$$\exists x \text{Corona}(x) \wedge \text{SullaTesta}(x, \text{Giovanni}).$$

$\exists x$ si legge "esiste un x tale che..." o anche "per qualche x ...".

Intuitivamente, la formula $\exists x P$ afferma che P è vera per almeno un oggetto x . Più precisamente, $\exists x P$ è vera in un dato modello se P è vera in *almeno una* interpretazione estesa che assegna a x un elemento del dominio. Questo significa che deve valere almeno una delle seguenti formule:

Riccardo Cuor di Leone è una corona \wedge Riccardo Cuor di Leone è sulla testa di Giovanni.

Re Giovanni è una corona \wedge Re Giovanni è sulla testa di Giovanni.

La gamba sinistra di Riccardo è una corona \wedge la gamba sinistra di Riccardo è sulla testa di Giovanni.

La gamba sinistra di Giovanni è una corona \wedge la gamba sinistra di Giovanni è sulla testa di Giovanni.

La corona è una corona \wedge la corona è sulla testa di Giovanni.

La quinta asserzione è vera nel modello, per cui anche la formula quantificata esistenzialmente lo è. Notate che, in base alla nostra definizione, la formula sarebbe vera anche in un

modello in cui Giovanni portasse due corone. Questo è del tutto compatibile con l'affermazione originale “Re Giovanni ha una corona sulla testa”.⁶

Proprio come \Rightarrow sembra essere il connettivo più naturale da usare insieme a \forall , \wedge è quello che meglio si presta a essere utilizzato con \exists . Usare \wedge come connettivo principale insieme a \forall portava, nell'esempio che abbiamo mostrato nel paragrafo precedente, a un'affermazione troppo forte; usare \Rightarrow con \exists conduce solitamente a formule troppo deboli. Considerate la seguente formula:

$$\exists x \text{ Corona}(x) \Rightarrow \text{SullaTesta}(x, \text{Giovanni}).$$

A prima vista, questa sembra essere un'espressione abbastanza ragionevole. Applicando la semantica, vediamo che la formula afferma che almeno una delle seguenti asserzioni è vera:

Riccardo Cuor di Leone è una corona \Rightarrow Riccardo Cuor di Leone è sulla testa di Giovanni;

Re Giovanni è una corona \Rightarrow Re Giovanni è sulla testa di Giovanni;

la gamba sinistra di Riccardo è una corona \Rightarrow la gamba sinistra di Riccardo è sulla testa di Giovanni;

e così via. Un'implicazione è vera se sono vere sia la sua premessa che la sua conclusione, oppure se la sua premessa è falsa. Così, se Riccardo Cuor di Leone non è una corona, allora la prima asserzione è vera e il quantificatore esistenziale è soddisfatto. In pratica, un'implicazione esistenzialmente quantificata è vera ogni volta che *un qualsiasi* oggetto non soddisfa la premessa; ne consegue che formule di questo tipo sono davvero molto deboli.

Quantificatori annidati

Spesso vorremo esprimere formule più complesse usando più quantificatori. Il caso più semplice si ha quando i quantificatori sono dello stesso tipo: per esempio, “i fratelli sono consanguinei” può essere scritto come:

$$\forall x \forall y \text{ Fratello}(x, y) \Rightarrow \text{Consanguineo}(x, y).$$

Quantificatori consecutivi dello stesso tipo possono essere scritti anche usando un solo quantificatore con più variabili. Per dire, per esempio, che la consanguineità è una relazione simmetrica, possiamo scrivere:

$$\forall x, y \text{ Consanguineo}(x, y) \Leftrightarrow \text{Consanguineo}(y, x).$$

In altri casi potrà capitare di mescolare quantificatori diversi. Dire che “ognuno ama qualcuno” significa che per ogni persona c'è una persona da essa amata:

$$\forall x \exists y \text{ Ama}(x, y).$$

D'altra parte, per affermare che “c'è qualcuno che è amato da tutti” scriviamo:

$$\exists y \forall x \text{ Ama}(x, y).$$

L'ordine con cui sono scritti i quantificatori, quindi, è molto importante. Per rendere la formula più chiara possiamo inserire parentesi: $\forall x (\exists y \text{ Ama}(x, y))$ afferma che *ognuno* ha una particolare proprietà, e in particolare il fatto che ama qualcuno. D'altra parte, $\exists y (\forall x \text{ Ama}(x, y))$ dice che *qualcuno* nel mondo ha una particolare caratteristica, e cioè che tutti lo amano.

Qualche confusione può sorgere quando due quantificatori sono applicati allo stesso nome di variabile: considerate la formula:

$$\forall x (\text{Corona}(x) \vee (\exists x \text{ Fratello}(\text{Riccardo}, x))).$$

⁶ Esiste una variante del quantificatore esistenziale, solitamente scritta \exists^1 o $\exists!$, che significa “esiste esattamente un”. Lo stesso significato può essere espresso usando l'uguaglianza.

Qui la x in $(\text{Fratello}(\text{Riccardo}, x)$ è quantificata *esistenzialmente*. La regola è che la variabile “appartiene” al quantificatore più interno che la menziona; quindi non sarà più soggetta a nessun’altra quantificazione. Un altro modo di spiegare cosa succede è dire che $\exists x \text{ Fratello}(\text{Riccardo}, x)$ è una formula che riguarda Riccardo (e dice che ha un fratello), e non riguarda x ; di conseguenza scrivere $\forall x$ al suo esterno non ha alcun effetto. In effetti, in modo del tutto equivalente avremmo potuto scrivere $\exists z \text{ Fratello}(\text{Riccardo}, z)$. Poiché tutto questo rappresenta una fonte di confusione, d’ora in poi useremo sempre nomi di variabile diversi con quantificatori annidati.

Connessioni tra \forall e \exists

I due quantificatori in effetti sono strettamente correlati attraverso la negazione. Dire che tutti odiano i broccoli è come dire che non esiste nessuno a cui piacciono, e viceversa:

$$\forall x \neg \text{Gradisce}(x, \text{Broccoli}) \quad \text{è equivalente a} \quad \neg \exists x \text{ Gradisce}(x, \text{Broccoli}).$$

Parimenti, dire che “a tutti piace il gelato” significa che non c’è proprio nessuno a cui non piaccia:

$$\forall x \text{ Gradisce}(x, \text{Gelato}) \quad \text{è equivalente a} \quad \neg \exists x \neg \text{Gradisce}(x, \text{Gelato}).$$

Dal momento che \forall è di fatto una congiunzione sull’universo degli oggetti, mentre \exists è una disgiunzione, non dovrebbe sorprendere che obbediscano alle leggi di De Morgan. Ecco le regole di De Morgan per le formule quantificate e non:

$$\begin{array}{ll} \neg \exists x \neg P & \equiv \forall x P \\ \neg \forall x P & \equiv \exists x \neg P \\ \forall x P & \equiv \neg \exists x \neg P \\ \exists x P & \equiv \neg \forall x \neg P \end{array} \quad \begin{array}{ll} \neg P \wedge \neg Q & \equiv \neg P \vee Q \\ \neg(P \wedge Q) & \equiv \neg P \vee \neg Q \\ P \wedge Q & \equiv \neg(\neg P \vee \neg Q) \\ P \vee Q & \equiv \neg(\neg P \wedge \neg Q). \end{array}$$

In definitiva non è strettamente necessario disporre sia di \forall che di \exists , proprio come non è indispensabile avere sia \wedge che \vee . In ogni caso la leggibilità è più importante della parsimonia, ragion per cui continueremo a usare entrambi i quantificatori.

8.2.7 Uguaglianza

simbolo di uguaglianza

La logica del primo ordine comprende un altro modo di costruire formule atomiche, senza usare predicati e termini. Possiamo utilizzare il **simbolo di uguaglianza** per affermare che due termini fanno riferimento allo stesso oggetto. Per esempio,

$$\text{Padre}(\text{Giovanni}) = \text{Enrico}$$

asserisce che l’oggetto a cui si riferisce $\text{Padre}(\text{Giovanni})$ e quello a cui si riferisce Enrico sono effettivamente lo stesso. Dato che ogni interpretazione fissa il riferimento di ogni termine, determinare la verità di una formula di uguaglianza consiste semplicemente nel verificare se i due termini si riferiscono allo stesso oggetto.

Il simbolo di uguaglianza si può usare per esprimere fatti riguardanti una data funzione, come fa l’esempio appena proposto nei confronti del simbolo Padre ; lo si può usare anche con la negazione per specificare che due termini non sono lo stesso oggetto. Per dire che Riccardo ha almeno due fratelli, possiamo scrivere:

$$\exists x, y \text{ Fratello}(x, \text{Riccardo}) \wedge \text{Fratello}(y, \text{Riccardo}) \wedge \neg(x = y).$$

La formula

$$\exists x, y \text{ Fratello}(x, \text{Riccardo}) \wedge \text{Fratello}(y, \text{Riccardo})$$

non ha lo stesso significato. In particolare, è vera nel modello della Figura 8.2, in cui Riccardo ha un fratello solo: considerate infatti l’interpretazione estesa in cui sia x che y sono as-

segnate a Re Giovanni. L'aggiunta di $\neg(x = y)$ permette di eliminare modelli simili. Come abbreviazione di $\neg(x = y)$ talvolta si usa la notazione $x \neq y$.

8.2.8 Semantica dei database

Continuando l'esempio del paragrafo precedente, supponiamo di credere che Riccardo abbia due fratelli, Giovanni e Goffredo.⁷ Potremmo scrivere

$$\text{Fratello}(\text{Giovanni}, \text{Riccardo}) \wedge \text{Fratello}(\text{Goffredo}, \text{Riccardo}), \quad (8.3)$$

ma così non rappresenteremmo in modo completo lo stato delle cose. Per prima cosa, questa asserzione è vera in un modello in cui Riccardo ha un solo fratello – dobbiamo aggiungere $\text{Giovanni} \neq \text{Goffredo}$. In secondo luogo, la formula non esclude modelli in cui Riccardo ha molti altri fratelli oltre a Giovanni e Goffredo. Quindi, la traduzione corretta di “i fratelli di Riccardo sono Giovanni e Goffredo” è la seguente:

$$\begin{aligned} &\text{Fratello}(\text{Giovanni}, \text{Riccardo}) \wedge \text{Fratello}(\text{Goffredo}, \text{Riccardo}) \wedge \text{Giovanni} \neq \text{Goffredo} \\ &\wedge \forall x \text{ Fratello}(x, \text{Riccardo}) \Rightarrow (x = \text{Giovanni} \vee x = \text{Goffredo}) \end{aligned}$$

Questa formula logica appare molto più complicata della frase corrispondente in linguaggio naturale. Ma se non siamo in grado di tradurre correttamente dal linguaggio naturale, il nostro sistema di ragionamento logico commetterà degli errori. Siamo in grado di costruire una semantica che consenta un'espressione logica più diretta?

Una proposta molto popolare nei sistemi di database è la seguente. Innanzitutto, imponiamo che ogni simbolo di costante faccia riferimento a un oggetto distinto – l'**ipotesi dei nomi unici**. In secondo luogo facciamo l'**ipotesi del mondo chiuso**, cioè che le formule atomiche non conosciute come vere sono false. Infine, invochiamo la **chiusura del dominio**, a indicare che ogni modello contiene un numero di elementi del dominio non superiore a quello degli elementi denominati dai simboli di costante. Sotto la semantica risultante, la formula dell'Equazione (8.3) stabilisce che i due fratelli di Riccardo sono Giovanni e Goffredo. Questa semantica è chiamata **semantica dei database** per distinguere la semantica standard della logica del primo ordine. La semantica dei database è utilizzata anche nei sistemi di programmazione logica, come è spiegato nel Paragrafo 9.4.4.

È istruttivo considerare l'insieme di tutti i possibili modelli sotto la semantica dei database per lo stesso caso mostrato nella Figura 8.4. La Figura 8.5 mostra alcuni dei modelli, da quello senza tuple che soddisfano la relazione a quello in cui tutte le tuple soddisfano la relazione. Con due oggetti ci sono quattro possibili tuple di due elementi, perciò ci sono $2^4 = 16$ di-

ipotesi dei nomi unici
ipotesi del mondo chiuso
chiusura del dominio
semantica dei database

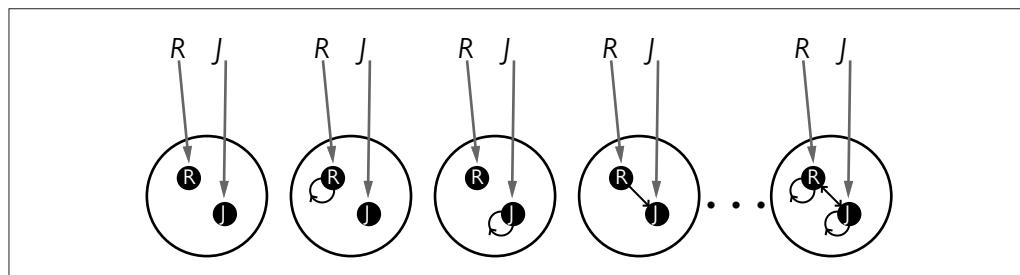


Figura 8.5 Alcuni membri dell'insieme di tutti i modelli per un linguaggio con due simboli di costante R e J e un solo simbolo di relazione binaria, sotto la semantica dei database. L'interpretazione dei simboli di costante è fissata ed esiste un oggetto distinto per ogni simbolo di costante.

⁷ In realtà ne aveva quattro, gli altri erano Guglielmo ed Enrico.

versi sottoinsiemi di tuple che possono soddisfare la relazione. Di conseguenza ci sono 16 modelli possibili in tutto, molti meno di quelli per la semantica del primo ordine standard, che sono infiniti. D'altra parte, la semantica dei database richiede una conoscenza definita di ciò che il mondo contiene.

Questo esempio mette in luce un punto importante: non esiste una semantica “corretta” per la logica. L'utilità di qualsiasi semantica proposta dipende da quanto concisa e intuitiva sia l'espressione dei tipi di conoscenza che vogliamo scrivere, e da quanto facile e naturale risulti sviluppare le regole di inferenza corrispondenti. La semantica dei database è utile soprattutto quando siamo certi dell'identità di tutti gli oggetti descritti nella base di conoscenza e quando abbiamo a disposizione tutti i fatti; in altri casi, non serve a molto. Per la parte restante di questo capitolo presupponiamo di utilizzare la semantica standard, evidenziando i casi in cui questa scelta porta a espressioni complicate.

8.3 Usare la logica del primo ordine

dominio

Ora che abbiamo definito un linguaggio logico espressivo, dobbiamo imparare a usarlo. In questo paragrafo forniremo delle formule di esempio in alcuni semplici **domini**. Nella rappresentazione della conoscenza, un dominio è semplicemente una parte del mondo riguardo a cui vogliamo esprimere appunto della conoscenza.

Cominciamo con una breve descrizione dell'interfaccia TELL/ASK per le basi di conoscenza del primo ordine. Esamineremo poi i domini delle relazioni familiari, dei numeri, degli insiemi, delle liste, e infine il mondo del wumpus. Il Paragrafo 8.4.2 presenta un esempio più articolato (i circuiti elettronici) mentre nel Capitolo 10 ci occuperemo dell'intero universo.

asserzione

8.3.1 Asserzioni e query nella logica del primo ordine

Proprio come nella logica proposizionale, per aggiungere formule alla base di conoscenza si usa TELL. Formule siffatte sono chiamate **asserzioni**. Per esempio, possiamo asserire che Giovanni è un re, Riccardo è una persona e che tutti i re sono persone:

```
TELL(KB, Re(Giovanni)).  
TELL(KB, Persona(Riccardo)).  
TELL(KB,  $\forall x \text{ Re}(x) \Rightarrow \text{Persona}(x)$ ).
```

Per interrogare la base di conoscenza si usa ASK: per esempio,

```
ASK(KB, Re(Giovanni))
```

restituisce *true*. Le domande poste con ASK prendono il nome di **query** (o interrogazioni) o anche **obiettivi**. In generale, ogni query che è conseguenza logica della base di conoscenza dovrebbe avere risposta affermativa. Per esempio, date le tre asserzioni precedenti, la query:

```
ASK(KB, Persona(Giovanni))
```

Dovrebbe anch'essa restituire *true*. Possiamo anche costruire query quantificate, come:

```
ASK(KB,  $\exists x \text{ Persona}(x)$ ).
```

La risposta a questa query è *true*, ma forse non è utile quanto vorremmo. In effetti, sarebbe come rispondere “Sì” alla domanda “Scusi, sa che ore sono?”. Se vogliamo conoscere quale valore di x rende vera la formula, ci servirà una funzione diversa, che chiamiamo ASKVARS:

```
ASKVARS(KB, Persona(x))
```

sostituzione
lista di legami

e che restituisce un flusso di risposte. In questo caso ci saranno due risposte: $\{x/Giovanni\}$ e $\{x/Riccardo\}$. Una risposta di questo tipo si chiama **sostituzione** o **lista di legami**. ASKVARS solitamente è riservata a basi di conoscenza costituite unicamente da clausole di Horn, per-

ché in tali basi di conoscenza ogni modo di rendere vera la query legherà le variabili a valori specifici. Non è questo il caso della logica del primo ordine: se a KB è stato detto solo $Re(Giovanni) \vee Re(Riccardo)$, non c'è un singolo legame con x che rende vera la query $\exists x Re(x)$, anche se tale query è effettivamente vera.

8.3.2 Il dominio della parentela

Il primo esempio che consideriamo è il dominio delle relazioni di parentela, che include fatti come “Elisabetta è la madre di Carlo” e “Carlo è il padre di William” e regole come “la nonna di un individuo è la madre di uno dei suoi genitori”.

Gli oggetti del nostro dominio sono tutte persone. I predicati unari comprendono *Maschio* e *Femmina*, tra gli altri. Le relazioni di parentela tra genitori, fratelli, coniugi e così via saranno rappresentate da predicati binari: *Genitore*, *Consanguineo*, *Fratello*, *Sorella*, *Figlio*, *Figlia*, *Progenie*, *Coniuge*, *Moglie*, *Marito*, *Nonno*, *Nipote*, *Cugino*, *Zia* e *Zio*. Per *Madre* e *Padre* useremo delle funzioni, dato che ogni persona ha esattamente un genitore maschio e uno femmina, dal punto di vista biologico (anche se potremmo introdurre funzioni aggiuntive per madri adottive, surrogate e così via).

Possiamo esaminare a turno ogni funzione e predicato e scrivere quello che sappiamo in relazione agli altri simboli. Per esempio, la madre di una persona è il suo genitore femmina:

$$\forall m, c \quad Madre(c) = m \Leftrightarrow Femmina(m) \wedge Genitore(m, c).$$

Il marito è il coniuge maschio di una persona:

$$\forall w, h \quad Marito(h, w) \Leftrightarrow Maschio(h) \wedge Coniuge(h, w).$$

Maschio e femmina sono categorie disgiunte:

$$\forall x \quad Maschio(x) \Leftrightarrow \neg Femmina(x).$$

Genitore e progenie sono relazioni inverse:

$$\forall p, c \quad Genitore(p, c) \Leftrightarrow Progenie(c, p).$$

Un nonno (o nonna) è un genitore di un genitore:

$$\forall g, c \quad Nonno(g, c) \Leftrightarrow \exists p \quad Genitore(g, p) \wedge Genitore(p, c).$$

Un consanguineo (corrispondente all'inglese *sibling*) è un altro figlio dei genitori di un individuo:

$$\forall x, y \quad Consanguineo(x, y) \Leftrightarrow x \neq y \wedge \exists p \quad Genitore(p, x) \wedge Genitore(p, y).$$

Potremmo andare avanti così per parecchie pagine, e l'Esercizio 8.KINS vi chiederà proprio questo.

Ognuna di queste formule può essere considerata un **assioma** nel dominio della parentela, come è spiegato nel Paragrafo 7.1. Normalmente gli assiomi sono associati ai domini puramente matematici (e in effetti ne vedremo tra poco alcuni relativi ai numeri), ma sono necessari in tutti i domini, rappresentando l'informazione base da cui si possono derivare conclusioni utili. I nostri assiomi di parentela sono anche **definizioni**: hanno tutti la forma $\forall x, y P(x, y) \Leftrightarrow \dots$ Gli assiomi definiscono la funzione *Madre* e i predicati *Marito*, *Maschio*, *Genitore*, *Nonno* e *Consanguineo* in termini di altri predicati. La “base” delle definizioni è costituita da un insieme di predicati (*Progenie*, *Femmina*, ecc.) che servono da riferimento per definire gli altri.

definizione

Questo è un modo naturale di costruire la rappresentazione di un dominio, ed è analogo al modo con cui si costruiscono i pacchetti software definendo nuove procedure sulla base di funzioni primitive di libreria. Notate che l'insieme dei predicati primitivi non dev'essere necessariamente unico: avremo potuto usare *Genitore* al posto di *Progenie*. In alcuni domini, come vedremo, non esiste neppure un insieme base chiaramente identificabile.

teorema

Non tutte le formule logiche riguardanti un dominio sono assiomi. Alcune sono **teoremi**, ovvero formule deducibili dagli assiomi stessi. Considerate per esempio l'asserzione che la relazione di consanguineità è simmetrica:

$$\forall x, y \quad Consanguineo(x, y) \Leftrightarrow Consanguineo(y, x).$$

Questo è un assioma o un teorema? In effetti, è un teorema che segue logicamente dall'assioma che definisce la consanguineità. Se chiedessimo alla base di conoscenza il valore di verità di questa formula, il risultato dovrebbe essere *true*.

Da un punto di vista puramente logico, una base di conoscenza necessita solo degli assiomi e non dei teoremi, dato che questi ultimi non estendono l'insieme di conclusioni che possono essere derivate. In pratica, tuttavia, i teoremi sono essenziali per ridurre il costo computazionale della derivazione di nuove formule. Senza teoremi un sistema dovrebbe ripartire ogni volta dai principî base, come un matematico che dovesse derivare nuovamente le regole dell'analisi per ogni nuovo problema.

Non tutti gli assiomi sono definizioni; alcuni si limitano a fornire informazioni generali aggiuntive riguardanti certi predicati. In effetti, alcuni predicati non hanno neppure una definizione completa perché non abbiamo abbastanza informazioni per caratterizzarli pienamente. Per esempio, non c'è un modo semplice di completare la formula:

$$\forall x \ Person(x) \Leftrightarrow \dots$$

Fortunatamente, la logica del primo ordine ci permette di usare il predicato *Persona* senza definirlo completamente. Invece di far ciò possiamo scrivere specifiche parziali delle caratteristiche possedute da ogni persona e di quelle che fanno sì che un individuo sia una persona:

$$\forall x \ Person(x) \Rightarrow \dots$$

$$\forall x \dots \Rightarrow Person(x).$$

Gli assiomi possono anche essere “semplici fatti” come *Maschio(Jim)* e *Coniuge(Jim, Laura)*. Tali fatti costituiscono la descrizione di specifiche istanze di problemi, permettendo di rispondere a domande altrettanto specifiche. Le risposte saranno quindi teoremi che seguono dagli assiomi. Spesso potrà capitare di attendere risposte che invece la base di conoscenza non è in grado di dare: per esempio, da *Coniuge(Jim, Laura)* ci si potrebbe aspettare (con le leggi di molti paesi) di inferire $\neg Coniuge(George, Laura)$; tuttavia per far questo gli assiomi che abbiamo fornito non sono sufficienti, anche se aggiungiamo $Jim \neq George$ come suggerito nel Paragrafo 8.2.8. Questo significa che manca un assioma: l'Esercizio 8.HILL vi chiederà di formularlo.

8.3.3 Numeri, insiemi e liste

numeri naturali

I numeri sono forse l'esempio più lampante di come sia possibile costruire una grande teoria partendo da un numero microscopico di assiomi. Qui descriveremo la teoria dei **numeri naturali**, ovvero degli interi non negativi. Utilizzeremo un predicato *NumNat* che sarà vero per i numeri naturali. Ci servirà poi un solo simbolo di costante, 0, e un solo simbolo di funzione, *S* (successore). Gli **assiomi di Peano** definiscono i numeri naturali e l'addizione.⁸ I numeri naturali sono definiti ricorsivamente:

$$NumNat(0)$$

$$\forall n \quad NumNat(n) \Rightarrow NumNat(S(n)) .$$

⁸ Gli assiomi di Peano includono anche il principio di induzione, che è una formula della logica del secondo ordine e non del primo. Spiegheremo l'importanza di questa distinzione nel Capitolo 9.

assiomi di Peano

Questo significa che 0 è un numero naturale e che, per ogni oggetto n , se n è un numero naturale allora lo è anche il suo successore $S(n)$. Ne consegue che i numeri naturali sono $0, S(0), S(S(0))$ e così via. Dobbiamo scrivere degli assiomi per restringere la funzione successore:

$$\forall n \ 0 \neq S(n) .$$

$$\forall m, n \ m \neq n \Rightarrow S(m) \neq S(n) .$$

Ora possiamo definire l'addizione in termini di funzione successore:

$$\forall m \ \text{NumNat}(m) \Rightarrow + (0, m) = m$$

$$\forall m, n \ \text{NumNat}(m) \wedge \text{NumNat}(n) \Rightarrow + (S(m), n) = S(+ (m, n)) .$$

Il primo di questi assiomi dice che aggiungere 0 a qualsiasi numero naturale m restituisce m stesso. Notate l'uso del simbolo di funzione binaria “+” nel termine $+(m, 0)$; normalmente lo avremmo scritto $m + 0$, usando la **notazione infissa**. Quella usata qui, invece, prende il nome di **notazione prefissa**. Per rendere più facile la lettura delle formule che riguardano i numeri permetteremo l'uso della notazione infissa: inoltre possiamo scrivere $S(n)$ come $n + 1$, e così il secondo assioma diventa:

$$\forall m, n \ \text{NumNat}(m) \wedge \text{NumNat}(n) \Rightarrow (m + 1) + n = (m + n) + 1 .$$

Quest'assioma riduce l'addizione all'applicazione ripetuta della funzione successore.

L'uso della notazione infissa è un esempio di **zucchero sintattico**, un'estensione o abbreviazione della sintassi standard che non modifica la semantica. Ogni formula che utilizza lo zucchero può essere “de-zuccherata” per produrre la formula equivalente nella sintassi ordinaria della logica del primo ordine. Un altro esempio è l'uso delle parentesi quadre anziché di quelle tonde per far capire più facilmente la corrispondenza tra le parentesi a sinistra e a destra. Un altro esempio è il raggruppamento dei quantificatori, che significa sostituire $\forall x \forall y P(x, y)$ con $\forall x, y P(x, y)$

notazione infissa
notazione prefissa

zucchero sintattico

A questo punto è molto semplice definire la moltiplicazione come addizioni ripetute, l'elevamento a potenza come moltiplicazioni ripetute, la divisione intera con il resto, i numeri primi e così via. Così, l'intera teoria dei numeri (inclusa la crittografia) può essere costruita partendo da una costante, una funzione, un predicato e quattro assiomi.

Anche il dominio degli **insiemi** è fondamentale per la matematica, come del resto per il ragionamento (tanto che è possibile costruire la teoria dei numeri partendo da quella degli insiemi). Vogliamo essere in grado di rappresentare singoli insiemi, tra cui l'insieme vuoto: ci serve un modo per costruire insiemi a partire da singoli elementi o da operazioni su altri insiemi. Vorremo sapere se un elemento fa parte o no di un insieme e avere il modo di distinguere gli insiemi da oggetti che non lo sono.

Come zucchero sintattico useremo il normale vocabolario della teoria degli insiemi. L'insieme vuoto è una costante, indicata con $\{\}$. C'è un solo predicato unario, Set , che è vero per gli insiemi. I predicati binari sono $x \in s$ (x è un elemento dell'insieme s) e $s_1 \subseteq s_2$ (s_1 è un sottoinsieme di s_2 , eventualmente uguale a s_2). Le funzioni binarie sono $s_1 \cap s_2$ (intersezione), $s_1 \cup s_2$ (unione) e $\text{Aggiungi}(x, s)$ (l'insieme risultante dall'aggiunta dell'elemento x a s). Un possibile insieme di assiomi è il seguente.

1. Gli unici insiemi sono quello vuoto e quelli costruiti aggiungendo qualcosa a un insieme:

$$\forall s \ \text{Set}(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2 \ \text{Set}(s_2) \wedge s = \text{Aggiungi}(x, s_2)) .$$

2. L'insieme vuoto non ha alcun elemento aggiunto; in altre parole non c'è modo di scomporre $\{\}$ in un insieme più piccolo e un elemento singolo:

$$\neg \exists x, s \ \text{Aggiungi}(x, s) = \{\} .$$

3. Aggiungere a un insieme un elemento che vi è già presente non ha alcun effetto:

$$\forall x, s \ x \in s \Leftrightarrow s = \text{Aggiungi}(x, s) .$$

4. I soli membri di un insieme sono gli elementi che vi sono stati aggiunti. Per esprimere questo ricorsivamente, diciamo che x è membro di s se e solo se s è uguale a un insieme s_2 a cui è stato aggiunto un elemento y , laddove y è lo stesso di x o x è un membro di s_2 :

$$\forall x, s \quad x \in s \Leftrightarrow \exists y, s_2 (s = \text{Aggiungi}(y, s_2) \wedge (x = y \vee x \in s_2)) .$$
5. Un insieme è un sottoinsieme di un altro insieme se e solo se tutti i suoi membri sono anche membri del secondo insieme:

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \quad x \in s_1 \Rightarrow x \in s_2) .$$
6. Due insiemi sono uguali se e solo se ognuno è un sottoinsieme dell’altro:

$$\forall s_1, s_2 \quad (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1) .$$
7. Un oggetto appartiene all’intersezione di due insiemi se e solo se è membro di entrambi:

$$\forall x, s_1, s_2 \quad x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2) .$$
8. Un oggetto appartiene all’unione di due insiemi se e solo se è membro dell’uno o dell’altro:

$$\forall x, s_1, s_2 \quad x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2) .$$

lista

Le **liste** sono simili agli insiemi, con la differenza che sono ordinate e che lo stesso elemento può apparire più volte. Per esse possiamo usare il vocabolario del Lisp: *Nil* è la lista costante priva di elementi; *Cons*, *Append*, *First* e *Rest* sono funzioni; *Find* è il predicato che verifica se un elemento è presente nella lista. *List* è un predicato che vale solo per le liste. Come nel caso degli insiemi, nelle formule logiche che coinvolgono le liste è frequente l’uso di zucchero sintattico. Così, la lista vuota si indica con []; il termine *Cons*(x , *Nil*) (ovvero la lista che contiene il solo elemento x) si scrive $[x]$. Una lista di più elementi, come $[A, B, C]$, corrisponde al termine annidato *Cons*(A , *Cons*(B , *Cons*(C , *Nil*))). Nell’Esercizio 8.LIST vi sarà richiesto di scrivere gli assiomi per le liste.

8.3.4 Il mondo del wumpus

Nel Capitolo 7 abbiamo già scritto alcuni assiomi in logica proposizionale per il mondo del wumpus. Gli assiomi in logica del primo ordine che forniremo ora sono molto più concisi e catturano in modo molto naturale esattamente ciò che vogliamo esprimere.

Ricorderete che l’agente nel mondo del wumpus riceve un vettore di percezioni composto da cinque elementi. La formula del primo ordine memorizzata nella base di conoscenza deve includere sia la percezione che l’istante nel quale è stata ricevuta; in caso contrario, l’agente farebbe una gran confusione tra le percezioni riferite a tempi diversi. Una tipica formula riguardante le percezioni sarà quindi:

$$\text{Percezione}([Fetore, Brezza, Scintillio, None, None], 5) .$$

Qui *Percezione* è un predicato binario mentre *Fetore* etc. sono costanti raccolte in una lista. Le azioni possibili possono essere rappresentate mediante termini logici:

$$Gira(\text{Destra}), Gira(\text{Sinistra}), Avanti, Scocca, Afferra, Esci.$$

Per determinare l’azione migliore, il programma agente esegue la query:

$$\text{ASKVARS}(KB, \text{MigliorAzione}(a, 5)) ,$$

che restituisce una lista di legami come $\{a/\text{Afferra}\}$. Il programma agente potrà quindi restituire *Afferra* come azione da effettuare. I dati relativi alle percezioni implicano, da soli, alcuni fatti riguardanti lo stato corrente. Per esempio:

$$\begin{aligned} &\forall t, s, g, m, c \quad \text{Percezione}([s, Brezza, g, m, c], t) \Rightarrow \text{Brezza}(t) \\ &\forall t, s, g, m, c \quad \text{Percezione}([s, None, g, m, c], t) \Rightarrow \neg\text{Brezza}(t) \\ &\forall t, s, b, m, c \quad \text{Percezione}([s, b, Scintillio, m, c], t) \Rightarrow \text{Scintillio}(t) \\ &\forall t, s, b, m, c \quad \text{Percezione}([s, b, None, m, c], t) \Rightarrow \neg\text{Scintillio}(t) \end{aligned}$$

e così via. Queste regole costituiscono una forma molto semplice di processo di ragionamento che prende appunto il nome di **percezione**, e che studieremo approfonditamente nel Capitolo 25 del Volume 2. Notate la quantificazione sul tempo t : nella logica proposizionale, avremmo avuto bisogno di una copia di ogni formula per ogni passo temporale.

Semplici comportamenti “reattivi” possono essere implementati da formule di implicazione quantificata. Per esempio, possiamo scrivere:

$$\forall t \text{ Scintillio}(t) \Rightarrow \text{MigliorAzione(Afferra, } t).$$

Date la percezione e le regole viste nei paragrafi precedenti, questo fornirebbe la conclusione desiderata $\text{MigliorAzione(Afferra, 5)}$: la cosa giusta da fare è raccogliere l’oro.

Abbiamo rappresentato gli input e gli output dell’agente; ora è giunto il momento di rappresentare l’ambiente stesso. Cominceremo con gli oggetti: i candidati più ovvi sono le stanze, i pozzi e il wumpus. Potremmo dare un nome a ogni locazione – $\text{Stanza}_{1,2}$ e così via – ma poi saremmo obbligati a esprimere il fatto che $\text{Stanza}_{1,2}$ e $\text{Stanza}_{1,3}$ sono adiacenti come informazione aggiuntiva, e per di più saremmo costretti a farlo per ogni coppia di stanze. In questo caso è meglio usare un termine complesso in cui la riga e la colonna della posizione compaiono come indici interi: per esempio possiamo usare una lista come $[1, 2]$. A questo punto possiamo definire le stanze adiacenti nel modo seguente:

$$\begin{aligned} \forall x, y, a, b \text{ Adiacente } ([x, y], [a, b]) &\Leftrightarrow \\ ((x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))) . \end{aligned}$$

Potremmo anche dare un nome diverso a ogni pozzo, ma anche questa scelta sarebbe inappropriata, benché la ragione sia diversa: infatti non c’è alcun motivo di distinguere un pozzo dall’altro.⁹ È più semplice usare un predicato unario *Pozzo* che sarà vero per le stanze che contengono pozzi. Infine, dato che esiste esattamente un wumpus, al posto di un predicato unario si può usare altrettanto bene una costante *Wumpus*.

La posizione dell’agente cambia nel tempo, per cui scriveremo $\text{Pos}(\text{Agente}, s, t)$ per indicare che l’agente si trova nella stanza s nell’istante t . Possiamo fissare il wumpus in una posizione specifica per sempre con $\forall t \text{ Pos}(\text{Wumpus}, [2, 2], t)$. Possiamo quindi dire che gli oggetti possono trovarsi in una sola posizione in un dato istante:

$$\forall x, s_1, s_2, t \text{ Pos}(x, s_1, t) \wedge \text{Pos}(x, s_2, t) \Rightarrow s_1 = s_2 .$$

Data la sua posizione corrente, l’agente può dedurre le proprietà della stanza dalle sue percezioni in quel momento. Per esempio, se l’agente percepisce una brezza, allora quella stanza è ventosa:

$$\forall s, t \text{ Pos}(\text{Agente}, s, t) \wedge \text{Brezza}(t) \Rightarrow \text{Ventosa}(s) .$$

È utile sapere che una *stanza* è ventosa, dato che sappiamo che i pozzi non possono muoversi: notate infatti che *Ventosa* non prevede un argomento che specifichi il tempo.

Avendo scoperto quali locazioni sono ventose o puzzolenti e, cosa molto importante, quali *non* lo sono, l’agente può dedurre la posizione dei pozzi e dello stesso wumpus. Mentre la logica proposizionale necessita di un assioma separato per ogni stanza (cfr. R_2 e R_3 nel Paragrafo 7.4.3), e richiederebbe un diverso insieme di assiomi per ogni forma geografica del mondo, la logica del primo ordine richiede soltanto un assioma:

$$\forall s \text{ Ventosa}(s) \Leftrightarrow \exists r \text{ Adiacente}(r, s) \wedge \text{Pozzo}(r) \tag{8.4}$$

⁹ In modo analogo la maggior parte delle persone non distingue tra di loro gli uccelli che, quando si fa inverno, migrano verso le regioni più calde. Un ornitologo che volesse studiarne gli schemi di migrazione, la percentuale di sopravvivenza e così via sarà *effettivamente* costretto a dare un nome a ogni uccello per mezzo di un anello intorno alla zampa, a causa della necessità di tener traccia di ogni individuo.

Similmente, nella logica del primo ordine possiamo quantificare sul tempo, perciò ci serve soltanto un assioma di stato successore per ciascun predicato, anziché una copia diversa per ogni passo temporale. Per esempio, l'assioma per la freccia (Equazione (7.2) nel Capitolo 7) diventa:

$$\forall t \text{ HaFreccia}(t+1) \Leftrightarrow (\text{HaFreccia}(t) \wedge \neg \text{Azione}(\text{Scocca}, t)).$$

Da queste due formule di esempio possiamo vedere che la formulazione in logica del primo ordine è altrettanto concisa della descrizione in linguaggio naturale fornita nel Capitolo 7. Siete invitati a costruire assiomi analoghi per la posizione e l'orientamento dell'agente; in questi casi, gli assiomi quantificano sullo spazio e sul tempo. Come nel caso della stima dello stato con la logica proposizionale, un agente può usare l'inferenza logica con assiomi di questo tipo per tenere traccia di aspetti del mondo che non sono osservati direttamente. Nel Capitolo 11 approfondiremo il tema degli assiomi di stato successore del primo ordine e del loro impiego per costruire piani.

8.4 Ingegneria della conoscenza nella logica del primo ordine

ingegneria della conoscenza

Il paragrafo precedente ha illustrato l'uso della logica del primo ordine per rappresentare la conoscenza in tre semplici domini. In questo paragrafo descriveremo il processo generale di costruzione di una base di conoscenza; tale processo prende il nome di **ingegneria della conoscenza** (*knowledge engineering*). Un ingegnere della conoscenza investiga un particolare dominio, impara quali concetti sono importanti e scrive una rappresentazione formale degli oggetti al suo interno e delle relazioni tra essi. Presenteremo il processo di ingegneria della conoscenza nel dominio dei circuiti elettronici. L'approccio scelto è adatto allo sviluppo di basi di conoscenza *di uso specifico*, il cui dominio è precisamente limitato e di cui si conosce già la gamma di possibili interrogazioni. Le basi di conoscenza *di uso generale*, che coprono un'ampia varietà dello scibile umano e servono a supportare compiti quali la comprensione del linguaggio naturale, saranno discusse nel Capitolo 10.

8.4.1 Il processo di ingegneria della conoscenza

Tra i diversi progetti di ingegneria della conoscenza ci sono grandi differenze di contenuto, dimensione e difficoltà, ma tutti includono i seguenti passi.

1. *Identificare le domande.* L'ingegnere della conoscenza deve delineare la gamma di domande a cui la KB dovrà rispondere e le categorie di fatti disponibili per ogni specifica istanza di problema. Per esempio, la base di conoscenza del wumpus dovrà essere in grado di scegliere le azioni da intraprendere o dovrà soltanto rispondere a domande riguardanti il contenuto dell'ambiente? I fatti percepiti dai sensori includeranno la posizione corrente? Il compito della KB determinerà quale conoscenza dev'essere rappresentata per collegare istanze del problema alle risposte. Questo passo è analogo al processo di descrizione PEAS nella progettazione degli agenti, visto nel Capitolo 2.
2. *Raccogliere la conoscenza rilevante.* L'ingegnere della conoscenza potrebbe già essere un esperto del dominio, o potrebbe lavorare insieme ad altri esperti per estrarre quello che sanno: quest'ultimo processo prende il nome di **acquisizione della conoscenza**. In questa fase non viene utilizzata una rappresentazione formale: lo scopo è comprendere il ruolo e l'estensione della base di conoscenza, determinata dal suo compito, e capire come funziona effettivamente il dominio. Nel mondo del wumpus, definito da un insieme di regole artificiali, è facile identificare la conoscenza rilevante (ma notate che la definizione di stanze adiacenti non era stata fornita esplicitamente sotto forma di regola del mondo). Nei domini

acquisizione della conoscenza

reali, il problema della rilevanza può risultare abbastanza difficile; un sistema dedicato alla simulazione di progetti VLSI, per esempio, potrebbe prendere in considerazione o no la presenza di capacità parassite o dell'effetto pelle.

3. *Definire un vocabolario di predicati, funzioni e costanti.* A questo punto occorre tradurre i concetti importanti del dominio in nomi di simboli logici. Nel far questo sorgono molti problemi di *stile*: come lo stile di programmazione, anche quello di ingegneria della conoscenza può avere un impatto significativo sul successo di un progetto. Per esempio, i pozzi devono essere rappresentati da oggetti o da predicati unari applicati alle stanze? L'orientamento dell'agente sarà una funzione o un predicato? La posizione del wumpus dipenderà dal tempo? Una volta fatte queste scelte, il risultato sarà un vocabolario che prende il nome di **ontologia** del dominio. Con la parola *ontologia* si intende una particolare teoria riguardante la natura dell'esistenza. Un'ontologia definisce le categorie di oggetti esistenti, ma non le loro specifiche caratteristiche e le relazioni tra esse.

ontologia

4. *Codificare la conoscenza generale riguardante il dominio.* L'ingegnere della conoscenza scrive gli assiomi per tutti i termini del vocabolario. Questo definisce precisamente (nei limiti del possibile) il significato di ogni termine, permettendo all'esperto del dominio di verificare il contenuto. Spesso questo passo rivela malintesi o lacune nel vocabolario, che dovranno essere colmate tornando al passo precedente e ripetendo il processo iterativamente.

5. *Codificare una descrizione dell'istanza del problema.* Se l'ontologia è ben definita, questo passo è semplice: si tratta di scrivere semplici formule atomiche che riguardano istanze di concetti che fanno già parte dell'ontologia. Per un agente logico le istanze dei problemi sono fornite dai sensori, mentre una base di conoscenza "incorporea" riceve le formule allo stesso modo con cui i programmi tradizionali ricevono dati di input.

6. *Interrogare la base di conoscenza e ottenere da essa risposte tramite la procedura di inferenza.* A questo punto si possono raccogliere i frutti del proprio lavoro: la procedura di inferenza, partendo dagli assiomi e dai fatti specifici del problema, sarà in grado di derivare i fatti che ci interessa conoscere. Evitiamo quindi la necessità di scrivere un algoritmo risolvente specifico per ogni applicazione.

7. *Debugging e valutazione della base di conoscenza.* Purtroppo le risposte saranno raramente corrette al primo tentativo. Per essere più precisi le risposte saranno quelle giuste per la base di conoscenza così com'è scritta, presumendo che la procedura di inferenza sia corretta, ma potranno essere diverse da quelle che ci saremmo aspettati. Se per esempio manca un assioma, alcune interrogazioni potranno non avere risposta: ne potrebbe scaturire un processo di debugging alquanto faticoso. Assiomi mancanti, o troppo deboli, possono essere identificati facilmente cercando interruzioni inattese nella catena di ragionamento. Per esempio, se la base di conoscenza include una regola diagnostica (cfr. Esercizio 8.WUMD) per trovare il wumpus,

$$\forall s \text{ Puzzolente}(s) \Rightarrow \text{Adiacente}(\text{Casa}(Wumpus), s)$$

invece del bicondizionale, l'agente non sarà mai capace di dimostrare l'*assenza* del wumpus. Gli assiomi scorretti possono essere identificati perché rappresentano asserzioni false riguardo il mondo. Per esempio, la formula:

$$\forall x \text{ NumeroDiGambe}(x, 4) \Rightarrow \text{Mammifero}(x)$$

è falsa per i rettili, gli anfibi e i tavoli. *La falsità di questa formula può essere determinata indipendentemente dal resto della base di conoscenza.* Al contrario, un tipico errore in un programma potrebbe avere quest'aspetto:

offset = posizione + 1.

È impossibile determinare se offset debba essere posizione o posizione+1 senza comprendere il contesto.



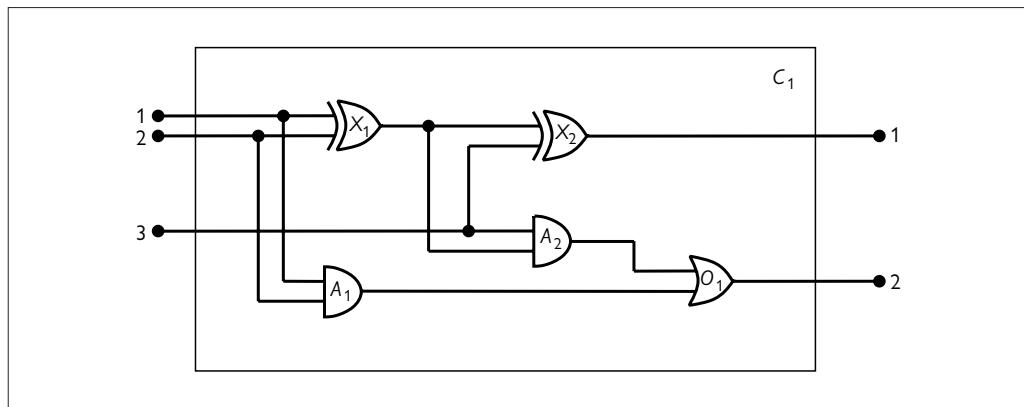


Figura 8.6 Un circuito digitale C_1 che dovrebbe essere un sommatore a un bit con riporto. I primi due input rappresentano gli addendi, il terzo è il riporto in ingresso. Il primo output è la somma, il secondo il riporto passato al sommatore successivo. Il circuito contiene due porte XOR, due porte AND e una porta OR.

Quando si arriva al punto in cui non vi sono più errori evidenti nella base di conoscenza, si è tentati di cantare vittoria, ma a meno che non sia davvero evidente l'assenza di errori, è meglio valutare formalmente il sistema eseguendo una serie di query di prova e misurando il numero di risposte corrette. Senza una misurazione oggettiva, è troppo facile autoconvincersi che il lavoro sia finito. Per comprendere appieno questo processo in sette passi, lo applicheremo ora a un esempio più esteso: il dominio dei circuiti elettronici.

8.4.2 Il dominio dei circuiti elettronici

Seguendo la metodologia a sette passi dell'ingegneria della conoscenza, svilupperemo un'ontologia e una base di conoscenza che ci permetteranno di ragionare sui circuiti elettrici del tipo mostrato nella Figura 8.6.

Identificare le domande

Esistono molte attività di ragionamento nel dominio dei circuiti digitali. Al livello di astrazione più alto, si può analizzarne la funzionalità: per esempio, il circuito nella Figura 8.6 esegue effettivamente un'addizione corretta? Se tutti gli input sono “alti”, qual è l'output della porta A_2 ? È anche interessante porre domande sulla struttura del circuito stesso. Per esempio, quali sono le porte collegate al primo morsetto di input? Il circuito contiene anelli di retroazione? In questo paragrafo, il compito della base di conoscenza sarà rispondere a domande come queste. Naturalmente ci sono livelli di analisi più dettagliati, che prendono in considerazione i ritardi temporali, l'area occupata dal circuito, il consumo di potenza, il costo di produzione e così via. Ognuno di questi livelli richiederebbe conoscenza aggiuntiva.

Raccogliere la conoscenza rilevante

Cosa sappiamo dei circuiti digitali? Per i nostri scopi, è sufficiente sapere che sono fatti di piste e di porte logiche. I segnali si propagano lungo le piste fino ai morsetti di input delle porte, che producono di conseguenza un segnale di output che viaggia su un'altra pista. Per determinare il valore dei segnali, dobbiamo conoscere il funzionamento delle porte logiche. Ne esistono quattro tipi diversi: le porte AND, OR e XOR hanno due morsetti di input, quelle NOT soltanto uno. Tutte le porte logiche hanno un solo morsetto di output. I circuiti, come le porte, hanno morsetti di input e di output.

Per ragionare sulla funzionalità di un circuito e sulla sua struttura non è necessario parlare esplicitamente delle piste, dei loro cammini e delle giunzioni che li uniscono. Basta conoscere i collegamenti tra i morsetti; è sufficiente dire che l'output di una porta è collegato all'input di un'altra senza menzionare ciò che li mette fisicamente in contatto. Altri fattori come la dimensione, la forma, il colore o il costo dei vari componenti sono irrilevanti ai fini della nostra analisi.

Se il nostro scopo andasse oltre la semplice verifica del funzionamento al livello delle porte logiche, la nostra ontologia sarebbe diversa. Per esempio, se fossimo interessati al debugging di circuiti difettosi, sarebbe stata una buona idea includere nell'ontologia le piste di rame, dato che una pista difettosa può corrompere il segnale che si propaga lungo di essa. Per risolvere gli errori di sincronizzazione avremmo dovuto prendere in considerazione i ritardi temporali introdotti dai transitori elettrici nelle porte. Se fossimo interessati alla costruzione di un prodotto in grado di essere venduto facilmente, sarebbe stato importante il costo del circuito e la sua velocità rispetto ai concorrenti sul mercato.

Definire un vocabolario

Ora sappiamo che vogliamo parlare di circuiti, morsetti, segnali e porte. Il passo successivo è scegliere le funzioni, i predicati e le costanti per rappresentare tali entità. Prima di tutto, dobbiamo poter distinguere una porta dalle altre. Ogni porta è rappresentata da un oggetto il cui nome è una costante, easseremo che è una porta con, per esempio, $Porta(X_1)$. Il comportamento di ogni porta è determinato dal suo tipo: una delle costanti AND , OR , XOR o NOT . Poiché una porta ha uno e un solo tipo, possiamo utilizzare una funzione: $Tipo(X_1) = XOR$. I circuiti, come le porte, sono identificati da un predicato: $Circuito(C_1)$.

Passiamo a considerare i morsetti, identificati dal predicato $Morsetto(x)$. Un circuito può avere uno o più morsetti di input e uno o più morsetti di output. Utilizziamo la funzione $In(1, X_1)$ per denotare il primo morsetto di input per il circuito X_1 . Una funzione simile, $Out(n, c)$, viene utilizzata per i morsetti di output. Il predicato $Arità(c, i, j)$ dice che il circuito c ha i morsetti di input e j morsetti di output. I collegamenti tra le porte logiche possono essere rappresentati con il predicato $Collegati$, che prende come argomenti due morsetti, come in $Collegati(Out(1, X_1), In(1, X_2))$.

Infine, dobbiamo sapere se un determinato segnale è acceso o spento (alto o basso, vero o falso e così via). Una possibilità è usare un predicato unario, $On(t)$, che varrà true quando il segnale a un morsetto è acceso. Questo, tuttavia, renderebbe alquanto difficile esprimere interrogazioni come “Quali sono i possibili valori dei segnali ai morsetti di output del circuito C_1 ?” Di conseguenza introduciamo come oggetti i due valori del segnale 1 e 0, che rappresentano rispettivamente “acceso” e “spento”, nonché una funzione $Segnale(t)$ che denota il valore del segnale per il morsetto t .

Codificare la conoscenza generale riguardante il dominio

Dover specificare poche regole generali, che possono essere espresse in modo chiaro e conciso, è segno di una buona ontologia. Di seguito sono riportati tutti gli assiomi che ci serviranno.

1. Se due morsetti sono collegati devono avere lo stesso segnale:

$$\forall t_1, t_2 \ Morsetto(t_1) \wedge Morsetto(t_2) \wedge Collegati(t_1, t_2) \Rightarrow Segnale(t_1) = Segnale(t_2).$$

2. A ogni morsetto il segnale vale 1 o 0:

$$\forall t \ Morsetto(t) \Rightarrow Segnale(t) = 1 \vee Segnale(t) = 0.$$

3. Essere collegati è commutativo:

$$\forall t_1, t_2 \ Collegati(t_1, t_2) \Leftrightarrow Collegati(t_2, t_1).$$

4. Ci sono quattro tipi di porte:

$$\forall g \ Porta(g) \wedge k = \text{Tipo}(g) \Rightarrow k = \text{AND} \vee k = \text{OR} \vee k = \text{XOR} \vee k = \text{NOT}.$$

5. L'output di una porta AND vale 0 se e solo se uno dei suoi input è 0:

$$\begin{aligned} \forall g \ Porta(g) \wedge \text{Tipo}(g) = \text{AND} \Rightarrow \\ \text{Segnale}(\text{Out}(1, g)) = 0 \Leftrightarrow \exists n \ \text{Segnale}(\text{In}(n, g)) = 0. \end{aligned}$$

6. L'output di una porta OR vale 1 se e solo se uno dei suoi input è 1:

$$\begin{aligned} \forall g \ Porta(g) \wedge \text{Tipo}(g) = \text{OR} \Rightarrow \\ \text{Segnale}(\text{Out}(1, g)) = 1 \Leftrightarrow \exists n \ \text{Segnale}(\text{In}(n, g)) = 1. \end{aligned}$$

7. L'output di una porta XOR vale 1 se e solo se i suoi input sono diversi:

$$\begin{aligned} \forall g \ Porta(g) \wedge \text{Tipo}(g) = \text{XOR} \Rightarrow \\ \text{Segnale}(\text{Out}(1, g)) = 1 \Leftrightarrow \text{Segnale}(\text{In}(1, g)) \neq \text{Segnale}(\text{In}(2, g)). \end{aligned}$$

8. L'output di una porta NOT è diverso dal suo input:

$$\begin{aligned} \forall g \ Porta(g) \wedge (\text{Tipo}(g) = \text{NOT}) \Rightarrow \\ \text{Segnale}(\text{Out}(1, g)) \neq \text{Segnale}(\text{In}(1, g)). \end{aligned}$$

9. Le porte (eccetto NOT) hanno due input e un output:

$$\begin{aligned} \forall g \ Porta(g) \wedge \text{Tipo}(g) = \text{NOT} \Rightarrow \text{Arità}(g, 1, 1). \\ \forall g \ Porta(g) \wedge k = \text{Tipo}(g) \wedge (k = \text{AND} \vee k = \text{OR} \vee k = \text{XOR}) \Rightarrow \text{Arità}(g, 2, 1). \end{aligned}$$

10. Un circuito ha morsetti, fino alla sua arità di input e output, e niente oltre la sua arità:

$$\begin{aligned} \forall c, i, j \ \text{Circuito}(c) \wedge \text{Arità}(c, i, j) \Rightarrow \\ \forall n \ (n \leq i \Rightarrow \text{Morsetto}(\text{In}(c, n))) \wedge (n > i \Rightarrow \text{In}(c, n) = \text{Niente}) \wedge \\ \forall n \ (n \leq j \Rightarrow \text{Morsetto}(\text{Out}(c, n))) \wedge (n > j \Rightarrow \text{Out}(c, n) = \text{Niente}). \end{aligned}$$

11. Porte, morsetti e segnali sono tutti distinti:

$$\begin{aligned} \forall g, t, s \ Porta(g) \wedge \text{Morsetto}(t) \wedge \text{Segnale}(s) \Rightarrow \\ g \neq t \wedge g \neq s \wedge t \neq s. \end{aligned}$$

12. Le porte sono circuiti:

$$\forall g \ Porta(g) \Rightarrow \text{Circuito}(g).$$

Codificare un'istanza specifica del problema

La seguente descrizione codifica il circuito mostrato nella Figura 8.6 come circuito C_1 . Per prima cosa indichiamo il tipo del circuito e delle porte che lo compongono:

$$\begin{aligned} \text{Circuito}(C_1) \wedge \text{Arità}(C_1, 3, 2) \\ \text{Porta}(X_1) \wedge \text{Tipo}(X_1) = \text{XOR} \\ \text{Porta}(X_2) \wedge \text{Tipo}(X_2) = \text{XOR} \\ \text{Porta}(A_1) \wedge \text{Tipo}(A_1) = \text{AND} \\ \text{Porta}(A_2) \wedge \text{Tipo}(A_2) = \text{AND} \\ \text{Porta}(O_1) \wedge \text{Tipo}(O_1) = \text{OR}. \end{aligned}$$

Quindi specifichiamo i collegamenti tra le porte:

$$\begin{array}{ll} \text{Collegati}(\text{Out}(1, X_1), \text{In}(1, X_2)) & \text{Collegati}(\text{In}(1, C_1), \text{In}(1, X_1)) \\ \text{Collegati}(\text{Out}(1, X_1), \text{In}(2, A_2)) & \text{Collegati}(\text{In}(1, C_1), \text{In}(1, A_1)) \\ \text{Collegati}(\text{Out}(1, A_2), \text{In}(1, O_1)) & \text{Collegati}(\text{In}(2, C_1), \text{In}(2, X_1)) \\ \text{Collegati}(\text{Out}(1, A_1), \text{In}(2, O_1)) & \text{Collegati}(\text{In}(2, C_1), \text{In}(2, A_1)) \\ \text{Collegati}(\text{Out}(1, X_2), \text{Out}(1, C_1)) & \text{Collegati}(\text{In}(3, C_1), \text{In}(2, X_2)) \\ \text{Collegati}(\text{Out}(1, O_1), \text{Out}(2, C_1)) & \text{Collegati}(\text{In}(3, C_1), \text{In}(1, A_2)). \end{array}$$

Interrogare la base di conoscenza

Quali combinazioni di input faranno sì che il primo output di C_1 (il bit di somma) valga 0 e il secondo output di C_1 (il bit di riporto) valga 1?

$$\begin{aligned} \exists i_1, i_2, i_3 \text{ Segnale}(In(1, C_1)) = i_1 \wedge \text{Segnale}(In(2, C_1)) = i_2 \wedge \text{Segnale}(In(3, C_1)) = i_3 \\ \wedge \text{Segnale}(Out(1, C_1)) = 0 \wedge \text{Segnale}(Out(2, C_1)) = 1. \end{aligned}$$

Le risposte saranno costituite da sostituzioni delle variabili i_1, i_2 e i_3 tali che la formula risultante sia conseguenza logica della base di conoscenza. ASKVARS ce ne fornirà tre:

$$\{i_1/1, i_2/1, i_3/0\} \quad \{i_1/1, i_2/0, i_3/1\} \quad \{i_1/0, i_2/1, i_3/1\}.$$

Quali sono i possibili insiemi di valori di tutti i morsetti del circuito sommatore?

$$\begin{aligned} \exists i_1, i_2, i_3, o_1, o_2 \text{ Segnale}(In(1, C_1)) = i_1 \wedge \text{Segnale}(In(2, C_1)) = i_2 \\ \wedge \text{Segnale}(In(3, C_1)) = i_3 \wedge \text{Segnale}(Out(1, C_1)) = o_1 \wedge \text{Segnale}(Out(2, C_1)) = o_2. \end{aligned}$$

Quest'ultima interrogazione restituirà una tabella input-output completa per il dispositivo, che può essere usata per verificare che sommi effettivamente in modo corretto i suoi input. Questo è un semplice esempio di **verifica dei circuiti**. Possiamo anche usare la definizione del circuito per costruire sistemi digitali più complessi, per i quali sarà possibile effettuare lo stesso tipo di verifica (cfr. Esercizio 8.ADDR). Questo sviluppo di una base di conoscenza strutturata è applicabile a una varietà di domini in cui è possibile esprimere i concetti più complessi partendo da quelli più semplici.

verifica dei circuiti

Fare il debugging della base di conoscenza

Possiamo perturbare la base di conoscenza in vari modi per vedere quale tipo di errori emerge nel suo comportamento. Per esempio, supponiamo di non aver letto il Paragrafo 8.2.8 e quindi di omettere l'asserzione che $1 \neq 0$. Supponiamo anche di determinare che il sistema non potrà calcolare alcun output del circuito, eccetto per i casi di input 000 e 110. Per localizzare il problema con precisione possiamo verificare gli output di ogni porta, chiedendo per esempio:

$$\exists i_1, i_2, o \text{ Segnale}(In(1, C_1)) = i_1 \wedge \text{Segnale}(In(2, C_1)) = i_2 \wedge \text{Segnale}(Out(1, X_1)) = o,$$

il che rivelerà che l'output di X_1 non è noto nel caso degli input 10 e 01. A questo punto basterà considerare gli assiomi delle porte XOR, così come si applicano a X_1 :

$$\text{Segnale}(Out(1, X_1)) = 1 \Leftrightarrow \text{Segnale}(In(1, X_1)) \neq \text{Segnale}(In(2, X_1)).$$

Se gli input valgono, poniamo, 1 e 0, questo si riduce a:

$$\text{Segnale}(Out(1, X_1)) = 1 \Leftrightarrow 1 \neq 0.$$

Ora il problema emerge in tutta evidenza: il sistema non è in grado di inferire che $\text{Segnale}(Out(1, X_1)) = 1$, così occorre dirgli che $1 \neq 0$.

8.5 Riepilogo

In questo capitolo abbiamo introdotto la **logica del primo ordine**, un linguaggio di rappresentazione molto più potente della logica proposizionale. I punti più importanti sono i seguenti.

- I linguaggi di rappresentazione della conoscenza dovrebbero essere dichiarativi, compositionali, espressivi, indipendenti dal contesto e non ambigui.
- Le logiche differiscono nei loro **impegni ontologici** e **impegni epistemologici**. Mentre la logica proposizionale si impegna solo sull'esistenza dei fatti, l'impegno della logica del primo ordine coinvolge l'esistenza di oggetti e relazioni e guadagna quindi potere espansivo, appropriato per domini come il mondo del wumpus e i circuiti elettronici.

- Sia la logica proposizionale sia quella del primo ordine condividono una difficoltà nel rappresentare proposizioni vaghe. Tale difficoltà limita l'applicabilità di queste logiche nei domini che richiedono giudizi personali, come la politica o la cucina.
- La sintassi della logica del primo ordine si basa su quella della logica proposizionale; aggiunge termini per rappresentare oggetti e dispone di quantificatori universali ed esistenziali per costruire asserzioni su tutti o alcuni dei possibili valori delle variabili quantificate.
- Un **mondo possibile** o **modello** per la logica del primo ordine include un insieme di oggetti e un'interpretazione che associa simboli di costante a oggetti, simboli di predicato a relazioni tra oggetti e simboli di funzione a funzioni su oggetti.
- Una **formula atomica** è vera soltanto quando la relazione indicata dal predicato è verificata tra gli oggetti a cui fanno riferimento i termini. Le **interpretazioni estese**, che associano le variabili dei quantificatori a oggetti del modello, definiscono il valore di verità di formule quantificate.
- Sviluppare una base di conoscenza in logica del primo ordine richiede un attento processo di analisi del dominio, scelta di un vocabolario e codifica degli assiomi necessari a supportare le inferenze desiderate.

Note storiche e bibliografiche

Sebbene la logica di Aristotele si occupasse di generalizzazioni sugli oggetti, non raggiungeva la potenza espressiva della logica del primo ordine. Una delle maggiori barriere al suo ulteriore sviluppo è stata l'eccessiva attenzione verso i predicati unari, con la conseguente esclusione di quelli ad aritma superiore. Il primo a trattare in modo sistematico le relazioni è stato Augustus De Morgan (1864), che ha fornito il seguente esempio per mostrare il tipo di inferenze che non potevano essere gestite dalla logica di Aristotele: "Tutti i cavalli sono animali; quindi la testa di un cavallo è la testa di un animale". Questa inferenza non è alla portata della logica aristotelica, perché ogni regola valida in grado di supportarla deve prima analizzare la formula con il predicato binario " x è la testa di y ". La logica delle relazioni è stata studiata approfonditamente da Charles Sanders Peirce (Peirce, 1870; Misak, 2004).

La nascita della logica del primo ordine si può far risalire all'introduzione dei quantificatori da parte di Gottlob Frege (1879) nel suo *Begriffschrift* (letteralmente "Scrittura dei concetti" o "Notazione concettuale"). La capacità di Frege di annidare i quantificatori rappresentò un grande passo avanti, ma la sua notazione era decisamente scomoda. La notazione usata oggi è dovuta principalmente a Giuseppe Peano (1889), ma la sua semantica è virtualmente identica a quella originale di Frege. In modo alquanto sorprendente,

dente, gli assiomi di Peano sono dovuti in gran parte a Grassmann (1861) e Dedekind (1888).

Leopold Löwenheim (1915) fornì un trattamento sistematico della teoria dei modelli per la logica del primo ordine: nel suo lavoro veniva trattato anche il simbolo di uguaglianza come parte integrante della logica. I risultati di Löwenheim furono estesi da Thoralf Skolem (1920). Alfred Tarski (1935, 1956), utilizzando la teoria degli insiemi, fornì una definizione esplicita di verità e di soddisfacimento in base alla teoria dei modelli nella logica del primo ordine.

John McCarthy (1958) è stato il più importante responsabile dell'adozione della logica del primo ordine come strumento per la costruzione di sistemi di IA. In seguito, l'applicazione della logica all'IA progredì significativamente con lo sviluppo da parte di Robinson (1965) della risoluzione, una procedura completa per inferenze del primo ordine. L'approccio logicista attecchi a Stanford: Cordell Green (1969a, 1969b) sviluppò un sistema di ragionamento del primo ordine, QA3, la qual cosa portò ai primi tentativi di costruzione di un robot logico presso lo SRI (Fikes e Nilsson, 1971). La logica del primo ordine fu applicata da Zohar Manna e Richard Waldinger (1971) al ragionamento sui programmi e più tardi, grazie a Michael Genesereth (1984), sui circuiti. In Europa, la programmazione logica (una forma ristretta di ragionamento

del primo ordine) conobbe uno sviluppo applicato all'analisi del linguaggio (Colmerauer *et al.*, 1973) e ai sistemi dichiarativi generali (Kowalski, 1974). La logica computazionale si guadagnò uno spazio importante anche a Edinburgo attraverso il progetto LCF (Logic for Computable Functions) (Gordon *et al.*, 1979). Tutti questi sviluppi saranno esaminati ulteriormente nei Capitoli 9 e 10.

Tra le applicazioni pratiche della logica del primo ordine vi è un sistema per valutare i requisiti di produzione per prodotti elettronici (Mannion, 2002), un sistema per il ragionamento riguardo le politiche di accesso ai file e la gestione dei diritti digitali (Halpern e Weissman, 2008), e un sistema per la composizione automatica di servizi web (McIlraith e Zeng, 2001).

Reazioni all'ipotesi di Whorf (Whorf, 1956) e il problema del linguaggio e del pensiero in generale appaiono in diversi libri recenti (Pullum, 1991; Pinker, 2003), inclusi due che sembrano in contraddizione tra loro: *Why the World Looks Different in Other Languages* (Deutscher, 2010) e *Why The World Looks the Same in Any Language* (McWhorter, 2014) (anche se entrambi gli autori concordano che vi siano differenze, ma ridotte). Un approccio (Gopnik e Glymour, 2002; Tenenbaum *et al.*, 2007) vede l'apprendimento del mondo da parte dei bambini come un processo analogo alla costruzione di teorie scientifiche. Esattamente come le predizioni di un algoritmo di apprendimento automatico dipendono fortemente dal vocabolario fornитогли, anche la formulazione di teorie da

parte del bambino dipende dall'ambiente linguistico in cui ha luogo l'apprendimento.

Ci sono diversi buoni testi introduttivi dedicati alla logica del primo ordine, tra cui quelli di personaggi chiave della storia della logica: Alfred Tarski (1941), Alonzo Church (1956) e W.V. Quine (1982), che è uno dei più leggibili. Enderton (1972) offre una prospettiva più matematica, mentre un trattamento molto formale della logica del primo ordine e di molti altri argomenti avanzati è fornito da Bell e Machover (1977). Manna e Waldinger (1985) hanno scritto un'introduzione di facile lettura alla logica dal punto di vista informatico, come Huth e Ryan (2004), che si sono concentrati sulla verifica dei programmi. Barwise ed Etchemendy (2002) hanno adottato un approccio simile a quello utilizzato qui. Smullyan (1995) presenta i risultati in modo conciso, usando i tableaux. Gallier (1986) fornisce un'esposizione matematica estremamente rigorosa della logica del primo ordine e una grande mole di materiale riguardante il suo uso nel ragionamento automatico. *Logical Foundations of Artificial Intelligence* (Genesereth e Nilsson, 1987) rappresenta sia una solida introduzione alla logica che un primo trattamento sistematico degli agenti logici con percezioni e azioni; inoltre ci sono due buoni manuali: van Benthem e ter Meulen (1997) e Robinson e Voronkov (2001). La rivista di riferimento nel campo della logica matematica pura è *Journal of Symbolic Logic*, mentre *Journal of Applied Logic* si occupa di argomenti più vicini a quelli dell'intelligenza artificiale.

Inferenza nella logica del primo ordine

- 9.1 Inferenza proposizionale e inferenza del primo ordine
- 9.2 Unificazione e inferenza del primo ordine
- 9.3 Concatenazione in avanti
- 9.4 Concatenazione all'indietro
- 9.5 Risoluzione
- 9.6 Riepilogo
- Note storiche e bibliografiche

Nel quale definiamo procedure efficaci per rispondere a domande poste con il linguaggio della logica del primo ordine.

In questo capitolo descriviamo algoritmi in grado di rispondere a qualsiasi domanda in logica del primo ordine per cui esiste una risposta. Il Paragrafo 9.1 introduce le regole di inferenza per i quantificatori e mostra come si può ridurre l'inferenza del primo ordine a quella proposizionale, sebbene questo comporti grandi costi. Il Paragrafo 9.2 spiega come si possa usare l'**unificazione** per costruire regole di inferenza che operano direttamente con formule del primo ordine. Discuteremo poi tre grandi famiglie di algoritmi di inferenza del primo ordine: la **concatenazione in avanti** (Paragrafo 9.3), la **concatenazione all'indietro** (Paragrafo 9.4) e i sistemi di **dimostrazione di teoremi basati sulla risoluzione** (Paragrafo 9.5).

9.1 Inferenza proposizionale e inferenza del primo ordine

Un modo per fare inferenza del primo ordine è quello di convertire la base di conoscenza del primo ordine in logica proposizionale e usare l'inferenza proposizionale, come sappiamo già fare. Un primo passo è quello di eliminare i quantificatori universali. Per esempio, supponiamo che la nostra base di conoscenza contenga il classico assioma popolare per cui tutti i re avidi sono malvagi:

$$\forall x \quad Re(x) \wedge Avido(x) \Rightarrow Malvagio(x).$$

Da qui possiamo inferire ognuna delle seguenti formule:

$$Re(Giovanni) \wedge Avido(Giovanni) \Rightarrow Malvagio(Giovanni)$$

$$Re(Riccardo) \wedge Avido(Riccardo) \Rightarrow Malvagio(Riccardo)$$

$$Re(Padre(Giovanni)) \wedge Avido(Padre(Giovanni)) \Rightarrow Malvagio(Padre(Giovanni))$$

⋮

istanziazione universale

In generale, la regola di **istanziazione universale** (in breve UI, dall'acronimo inglese) afferma che possiamo inferire tutte le formule ottenute sostituendo un **termine ground** (cioè privo di variabili) a una variabile quantificata universalmente.¹

Per scrivere formalmente la regola di inferenza useremo la nozione di **sostituzione** introdotta nel Paragrafo 8.3. Indichiamo con $\text{SUBST}(\theta, \alpha)$ il risultato dell'applicazione della sostituzione θ alla formula α . La regola allora si scrive:

$$\frac{\forall v \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

istanziazione esistenziale

per ogni variabile v e termine ground g . Le tre formule riportate in precedenza, per esempio, sono ottenute con le sostituzioni $\{x/Giovanni\}$, $\{x/Riccardo\}$ e $\{x/Padre(Giovanni)\}$.

Analogamente, la regola di **istanziazione esistenziale** sostituisce una variabile quantificata esistenzialmente con un unico *nuovo simbolo di costante*: per ogni formula α , variabile v e simbolo di costante k che non appare da nessun'altra parte nella base di conoscenza,

$$\frac{\exists v \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

Per esempio, dalla formula:

$$\exists x \text{ Corona}(x) \wedge \text{SullaTesta}(x, \text{Giovanni})$$

possiamo inferire la formula:

$$\text{Corona}(C_1) \wedge \text{SullaTesta}(C_1, \text{Giovanni})$$

A patto che C_1 non appaia in alcun altro punto della base di conoscenza. Sostanzialmente la formula quantificata esistenzialmente afferma che esiste un qualche oggetto che soddisfa una condizione, e l'applicazione della regola di istanziazione esistenziale si limita a dare un nome a tale oggetto; naturalmente quel nome non deve già appartenere a un altro oggetto. La matematica ci offre un altro esempio: supponiamo di scoprire un numero un po' più grande di 2,71828 che soddisfa l'equazione $d(x^y)/dy = x^y$ per x . Possiamo dare a questo numero il nome e , ma sarebbe un errore assegnargli quello di un oggetto preesistente, come π . In logica, il nuovo nome viene chiamato **costante di Skolem**.

costante di Skolem

Laddove la UI può essere applicata più volte allo stesso assioma per produrre conseguenze diverse, con l'istanziazione esistenziale basta una sola applicazione, dopodiché la formula quantificata esistenzialmente può essere scartata. Per esempio, non abbiamo più bisogno della formula $\exists x \text{ Uccide}(x, \text{Vittima})$, una volta che abbiamo aggiunto la formula $\text{Uccide}(\text{Assassino}, \text{Vittima})$.

9.1.1 Riduzione all'inferenza proposizionale

Mostriamo ora come convertire qualsiasi base di conoscenza del primo ordine in una base di conoscenza proposizionale. Innanzitutto, proprio come una formula quantificata esistenzialmente può essere rimpiazzata da una sua istanza, così una formula quantificata universalmente può essere sostituita dall'insieme di *tutte le possibili* istanze. Supponete per esempio che la base di conoscenza contenga solo le formule:

$$\begin{aligned} &\forall x \text{ Re}(x) \wedge \text{Avido}(x) \Rightarrow \text{Malvagio}(x) \\ &\text{Re}(\text{Giovanni}) \\ &\text{Avido}(\text{Giovanni}) \\ &\text{Fratello}(\text{Riccardo}, \text{Giovanni}) \end{aligned} \tag{9.1}$$

¹ Non confondete queste sostituzioni con le interpretazioni estese usate per definire la semantica dei quantificatori nel Paragrafo 8.2.6. La sostituzione rimpiazza una variabile con un termine (sintattico) per produrre una nuova formula, laddove un'interpretazione mette in corrispondenza variabili e oggetti del dominio.

e che gli unici oggetti siano *Giovanni* e *Riccardo*. Applichiamo la UI alla prima formula usando tutte le possibili sostituzioni, $\{x/\text{Giovanni}\}$ e $\{x/\text{Riccardo}\}$. Otteniamo:

$$\text{Re}(\text{Giovanni}) \wedge \text{Avido}(\text{Giovanni}) \Rightarrow \text{Malvagio}(\text{Giovanni})$$

$$\text{Re}(\text{Riccardo}) \wedge \text{Avido}(\text{Riccardo}) \Rightarrow \text{Malvagio}(\text{Riccardo}) .$$

Ora sostituiamo le formule atomiche ground, come $\text{Re}(\text{Giovanni})$ o $\text{Avido}(\text{Giovanni})$, con simboli proposizionali come ReGiovanni . Infine, applichiamo uno qualsiasi degli algoritmi proposizionali del Capitolo 7 per ottenere conclusioni come *Giovanni* È *Malvagio*, che è equivalente a *Malvagio(Giovanni)*.

Questa tecnica di **proposizionalizzazione** può essere applicata in modo assolutamente generale, come dimostriamo nel Paragrafo 9.5. Tuttavia, c'è un problema: quando la base di conoscenza include un simbolo di funzione, l'insieme di possibili sostituzioni di termini ground diventa infinito! Per esempio, se la base di conoscenza contiene il simbolo *Padre*, si possono costruire infiniti termini annidati come $\text{Padre}(\text{Padre}(\text{Padre}(\text{Padre}(\text{Giovanni}))))$.

proposizionalizzazione

Fortunatamente, il famoso teorema di Jacques Herbrand (1930) ci assicura che se una formula segue logicamente dalla base di conoscenza originale, quella del primo ordine, allora ci sarà una dimostrazione che coinvolge solo un sottoinsieme *finito* della base proposizionalizzata. Dato che in un qualsiasi sottoinsieme finito la profondità di annidamento dei termini deve essere limitata, possiamo generare prima tutte le istanziazioni con simboli di costante (*Riccardo* e *Giovanni*), poi tutti i termini di profondità 1 ($\text{Padre}(\text{Riccardo})$ e $\text{Padre}(\text{Giovanni})$), quindi tutti i termini di profondità 2, e così via finché non saremo in grado di costruire la dimostrazione proposizionale della formula che è conseguenza logica.

Il metodo che abbiamo delineato per effettuare inferenze in logica del primo ordine attraverso la proposizionalizzazione è **completo**: ogni formula che segue logicamente può essere dimostrata. Questo è un risultato notevole, dato che lo spazio di tutti i possibili modelli è infinito. D'altra parte, finché la dimostrazione non è terminata non sappiamo se la formula è *effettivamente* conseguenza logica! Che cosa succede quando *non* lo è? Possiamo determinarlo? Nella logica del primo ordine, questo risulta impossibile; il nostro procedimento continuerà a generare termini sempre più profondi, ma non sapremo mai se ci troviamo incastriati in un ciclo senza speranza o se la dimostrazione sta per terminare. Questo problema ricorda molto da vicino la terminazione delle macchine di Turing: lo stesso Turing (1936) e Alonzo Church (1936) hanno dimostrato, in modi alquanto differenti, che non c'è alcun modo di sfuggirvi. *Il problema dalla conseguenza logica, per la logica del primo ordine, è semi-decidibile: questo significa che esistono algoritmi che rispondono affermativamente per ogni formula che è conseguenza logica, ma nessun algoritmo potrà rispondere negativamente per ogni formula che non è conseguenza logica.*



9.2 Unificazione e inferenza del primo ordine

I lettori più attenti avranno notato che l'approccio della proposizionalizzazione genera molte istanziazioni non necessarie di formule quantificate universalmente. Sarebbe meglio avere un approccio con un'unica regola che ricava che $\{x/\text{Giovanni}\}$ risolve l'interrogazione *Malvagio(x)* come segue: data la regola che i re avidi sono malvagi, si trova un x tale che x è un re e x è avido, dopodiché si inferisce che questo x è malvagio. Più in generale, se esiste una sostituzione θ che rende ognuno dei congiunti della premessa dell'implicazione identico a una formula già presente nella base di conoscenza, allora dopo aver applicato θ possiamo asserirne anche la conclusione. In questo caso, la sostituzione $\theta = \{x/\text{Giovanni}\}$ raggiunge lo scopo. Ora supponiamo che, invece di limitarci a dire che *Avido(Giovanni)*, sappiamo che *tutti* sono avidi:

$$\forall y \text{ Avido}(y) . \tag{9.2}$$

Allora vorremmo continuare a essere in grado di concludere che *Malvagio(Giovanni)*, perché sappiamo che Giovanni è un re (fatto noto) e che è avido (perché lo sono tutti). Per far questo dobbiamo trovare una sostituzione sia per le variabili nell'implicazione che per quelle nelle formule presenti nella base di conoscenza. In questo caso, applicare la sostituzione $\{x/Giovanni, y/Giovanni\}$ alle premesse dell'implicazione *Re(x)* e *Avido(x)* nonché alle formule della base di conoscenza *Re(Giovanni)* e *Avido(y)* le renderà identiche. Di conseguenza potremo inferire il conseguente dell'implicazione.

Modus Ponens generalizzato

Questo processo di inferenza può essere espresso da una singola regola chiamata **Modus Ponens generalizzato**:² per le formule atomiche p_i , p'_i , e q , ove ci sia una sostituzione θ tale che $\text{SUBST}(\theta, p'_i) = \text{SUBST}(\theta, p_i)$ per tutti gli i ,

$$\frac{p'_1, p'_2, \dots, p'_n, (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}.$$

Questa regola ha $n + 1$ premesse: le n formule atomiche p'_i e la singola implicazione. La conclusione è il risultato dell'applicazione della sostituzione θ al conseguente q . Nel nostro esempio:

$$\begin{array}{ll} p'_1 \text{ è } \text{Re}(Giovanni) & p_1 \text{ è } \text{Re}(x) \\ p'_2 \text{ è } \text{Avido}(y) & p_2 \text{ è } \text{Avido}(x) \\ \theta \in \{x/Giovanni, y/Giovanni\} & q \text{ è } \text{Malvagio}(x) \\ \text{SUBST}(\theta, q) \text{ è } \text{Malvagio}(Giovanni). & \end{array}$$

È facile dimostrare che il Modus Ponens generalizzato è una regola di inferenza corretta. Prima di tutto osserviamo che, per ogni formula p (le cui variabili si assumono universalmente quantificate) e per ogni sostituzione θ ,

$$p \models \text{SUBST}(\theta, p).$$

è vero per istanziazione universale. È vero in particolare per una θ che soddisfa le condizioni del Modus Ponens generalizzato. Così, da p'_1, \dots, p'_n possiamo inferire:

$$\text{SUBST}(\theta, p'_1) \wedge \dots \wedge \text{SUBST}(\theta, p'_n)$$

e dall'implicazione $p_1 \wedge \dots \wedge p_n \Rightarrow q$ possiamo inferire:

$$\text{SUBST}(\theta, p_1) \wedge \dots \wedge \text{SUBST}(\theta, p_n) \Rightarrow \text{SUBST}(\theta, q).$$

Ora, nel Modus Ponens generalizzato, θ è definita in modo tale che $\text{SUBST}(\theta, p'_1) = \text{SUBST}(\theta, p_1)$ per tutti gli i ; ne consegue che la prima di queste due formule corrisponde esattamente alla premessa della seconda. Quindi $\text{SUBST}(\theta, q)$ segue per Modus Ponens.

lifting

Il Modus Ponens generalizzato è ottenuto dal Modus Ponens che già conosciamo attraverso un processo noto come **lifting** (letteralmente “sollevamento”). Il termine deriva dal fatto che abbiamo “sollevato” il Modus Ponens, portandolo dalla logica proposizionale, ground (senza variabili), a quella del primo ordine. Nel resto del capitolo vedremo che attraverso il lifting si possono sviluppare versioni per la logica del primo ordine della concatenazione in avanti, di quella all'indietro e dell'algoritmo di risoluzione, concetti che abbiamo già presentato nel Capitolo 7. Il vantaggio principale dell'uso di regole di inferenza “sollevate”, rispetto alla proposizionalizzazione, sta nel fatto che le prime effettuano solo le sostituzioni effettivamente necessarie per portare avanti le inferenze.

² Il Modus Ponens generalizzato è più generale del Modus Ponens (cfr. Paragrafo 7.5.1) nel senso che i fatti noti e le premesse dell'implicazione devono corrispondere soltanto tramite una sostituzione e non esattamente. D'altra parte, il Modus Ponens consente come premessa qualsiasi formula α e non soltanto una congiunzione di formule atomiche.

9.2.1 Unificazione

Le regole di inferenza ottenute attraverso il lifting richiedono di trovare sostituzioni che rendono identiche espressioni logiche diverse. Questo processo è chiamato **unificazione** ed è un componente chiave di tutti gli algoritmi di inferenza del primo ordine. L'algoritmo UNIFY prende due formule e, se esiste, restituisce un loro **unificatore** (una sostituzione):

$$\text{UNIFY}(p, q) = \theta \text{ dove } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q).$$

Vediamo qualche esempio di come si dovrebbe comportare UNIFY. Supponiamo di avere una query *Conosce(Giovanni, x)*: chi conosce Giovanni? Si possono trovare delle risposte a quest'interrogazione cercando tutte le formule nella base di conoscenza che si unificano con *Conosce(Giovanni, x)*. Ecco il risultato dell'unificazione con quattro formule diverse che potrebbero far parte della base di conoscenza:

$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(\text{Giovanni}, \text{Giacomina})) = \{x/\text{Giacomina}\}$$

$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(y, \text{Guglielmo})) = \{x/\text{Guglielmo}, y/\text{Giovanni}\}$$

$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(y, \text{Madre}(y))) = \{y/\text{Giovanni}, x/\text{Madre}(\text{Giovanni})\}$$

$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(x, \text{Elisabetta})) = \text{fallimento}.$$

L'ultima unificazione fallisce, perché *x* non può assumere contemporaneamente i valori *Giovanni* ed *Elisabetta*. Ricordate però che *Conosce(x, Elisabetta)* significa “tutti conoscono Elisabetta”, per cui *dovremmo* essere in grado di inferire che anche Giovanni la conosce. Il problema si verifica unicamente perché le due formule usano lo stesso nome per la variabile *x*. Tutto ciò può essere evitato ridenominando le variabili di una delle formule per evitare collisioni, un'operazione che prende il nome di **standardizzazione separata** (*standardizing apart*). Potremmo per esempio ridenominare la *x* di *Conosce(x, Elisabetta)* in *z₁₇* (un nuovo nome) senza modificare il suo significato. Ora l'unificazione funzionerà:

$$\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(z_{17}, \text{Elisabetta})) = \{x/\text{Elisabetta}, z_{17}/\text{Giovanni}\}.$$

L'Esercizio 9.STAN esaminerà più a fondo questo tipo di operazione.

C'è però un'ulteriore complicazione: abbiamo detto che UNIFY dovrebbe restituire una sostituzione tale che i due argomenti appaiano identici; ma potrebbe esistere anche più di un unificatore. Per esempio, $\text{UNIFY}(\text{Conosce}(\text{Giovanni}, x), \text{Conosce}(y, z))$ potrebbe restituire $\{y/\text{Giovanni}, x/z\}$ o $\{y/\text{Giovanni}, x/\text{Giovanni}, z/\text{Giovanni}\}$. Il primo unificatore dà come risultato *Conosce(Giovanni, z)*, mentre il secondo dà *Conosce(Giovanni, Giovanni)*. Il secondo risultato potrebbe essere anche ottenuto dal primo con la sostituzione aggiuntiva $\{z/\text{Giovanni}\}$; in questo caso diciamo che il primo unificatore è *più generale* del secondo, dato che impone meno restrizioni sul valore delle variabili.

Per ogni coppia di espressioni unificabili, esiste un singolo **unificatore più generale** (o MGU, dall'acronimo inglese), distinto da tutti gli altri a meno di ridenominazioni e sostituzioni di variabili. Per esempio, $\{x/\text{Giovanni}\}$ e $\{y/\text{Giovanni}\}$ sono considerati equivalenti, come $\{x/\text{Giovanni}, y/\text{Giovanni}\}$ e $\{x/\text{Giovanni}, y/x\}$.

La Figura 9.1 presenta un algoritmo per il calcolo degli MGU. Il processo è semplice: si esplorano simultaneamente le due espressioni “fianco a fianco” in modo ricorsivo, costruendo contemporaneamente un unificatore, e restituendo un risultato di fallimento se si incontrano nella loro struttura due punti non corrispondenti. L'algoritmo include un passo di costo computazionale elevato: quando si cerca una corrispondenza tra una variabile e un termine complesso, è necessario controllare se la variabile è presente all'interno dello stesso termine; in tal caso la corrispondenza fallisce perché non è possibile costruire un unificatore consistente. Per esempio, *S(x)* non si può unificare con *S(S(x))*. Questo cosiddetto **controllo di occorrenza** fa sì che la complessità dell'intero algoritmo risulti quadratica nella dimensione delle espressioni da unificare. Alcuni sistemi, tra cui molti sistemi di programmazione

unificazione

unificatore

standardizzazione separata

unificatore più generale

controllo di occorrenza

```

function UNIFY( $x, y, \theta = \text{vuoto}$ ) returns una sostituzione che rende  $x$  e  $y$  identici, o fallimento
  if  $\theta = \text{fallimento}$  then return fallimento
  else if  $x = y$  then return  $\theta$ 
  else if VARIABILE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
  else if VARIABILE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
  else if COMPOSTA?( $x$ ) and COMPOSTA?( $y$ ) then
    return UNIFY(ARGS( $x$ ), ARGS( $y$ ), UNIFY(OP( $x$ ), OP( $y$ ),  $\theta$ ))
  else if LISTA?( $x$ ) and LISTA?( $y$ ) then
    return UNIFY(RESTO( $x$ ), RESTO( $y$ ), UNIFY(PRIMO( $x$ ), PRIMO( $y$ ),  $\theta$ ))
  else return fallimento

function UNIFY-VAR( $var, x, \theta$ ) returns una sostituzione
  if  $\{var/val\} \in \theta$  per qualche  $val$  then return UNIFY( $val, x, \theta$ )
  else if  $\{x/val\} \in \theta$  per qualche  $val$  then return UNIFY( $var, val, \theta$ )
  else if CONTROLLA-OCCORRENZA?( $var, x$ ) then return fallimento
  else return aggiungi  $\{var/x\}$  a  $\theta$ 

```

Figura 9.1 L'algoritmo di unificazione. Gli argomenti x e y possono essere qualsiasi espressione: una costante o variabile, oppure un'espressione composta come una formula complessa o un termine, o una lista di espressioni. L'argomento θ è una sostituzione, inizialmente la sostituzione vuota, ma con l'aggiunta di coppie $\{var/val\}$ effettuata con la ricorsione sugli input, confrontando l'espressione elemento per elemento. In un'espressione composta, come $F(A, B)$, OP(x) restituisce il simbolo di funzione F e ARGS(x) restituisce la lista di argomenti (A, B).

logica, si limitano a omettere il controllo di occorrenza affidando all'utente l'onere di evitare di effettuare inferenze non corrette. Altri sistemi utilizzano algoritmi di unificazione più complessi che hanno complessità lineare.

9.2.2 Memorizzazione e recupero di informazioni

Alla base delle funzioni TELL, ASK e ASKVARS usate per informare e interrogare una base di conoscenza, ci sono le funzioni primitive STORE e FETCH. STORE(s) memorizza la formula s nella base di conoscenza, FETCH(q) restituisce tutti gli unificatori tali che la query q unifichi con una formula presente nella base di conoscenza. Il problema che abbiamo proposto precedentemente, di trovare tutti i fatti che unificano con *Conosce(Giovanni, x)*, è un esempio di FETCH.

Il modo più semplice di implementare STORE e FETCH è inserire tutti i fatti in una lunga lista e cercare di unificare ogni query con ogni elemento della lista. Un processo simile è inefficiente ma funziona. Ora accenneremo sommariamente alle tecniche utilizzate per rendere il recupero delle informazioni più efficiente.

Possiamo rendere FETCH più efficiente assicurandoci che cerchi di effettuare l'unificazione solo quando le formule assicurano che esista almeno una *qualche possibilità* di riuscita. Per esempio, è del tutto inutile cercare di unificare *Conosce(Giovanni, x)* con *Fratello(Riccardo, Giovanni)*. Possiamo evitare tali unificazioni **indicizzando** i fatti nella base di conoscenza. Un semplice schema chiamato **indicizzazione dei predicatori** pone tutti i fatti *Conosce* in un *bucket* (termine tecnico che letteralmente significa “secchio”) e tutti i fatti *Fratello* in un altro. I bucket possono essere memorizzati in una tabella di hash per un accesso efficiente.

L'indicizzazione dei predicatori è utile quando ci sono molti simboli di predicato ma solo poche formule per ognuno di essi. Talvolta, tuttavia, un predicato ha molte formule. Sup-

poniamo per esempio che il fisco voglia tener traccia di chi lavora per ogni azienda con il predicato $Dipendente(x, y)$. Questo darebbe origine a un bucket molto esteso, con milioni di datori di lavoro e decine di milioni di dipendenti, e rispondere a una semplice query come $Dipendente(x, Riccardo)$ attraverso l'indicizzazione dei prediciati richiederebbe la scansione dell'intero bucket.

Per questa particolare interrogazione, un grande aiuto sarebbe rappresentato dall'indicizzazione dei fatti mediante sia il predicato che il secondo argomento, magari usando una chiave di hash combinata. A quel punto si potrebbe semplicemente costruire la chiave dalla query ed estrarre esattamente i fatti che unificano con la query stessa. Per altri tipi di interrogazione, come $Dipendente(IBM, y)$, dovremmo indicizzare i fatti combinando il predicato e il primo argomento. I fatti quindi possono essere memorizzati in base a chiavi d'accesso multiple, rendendoli così istantaneamente disponibili alle varie query con cui potrebbero unificare.

Data una formula da memorizzare, è possibile costruire indici per *tutte le possibili* query che dovranno unificare con essa. Per il fatto $Dipendente(IBM, Riccardo)$, le interrogazioni sono:

$Dipendente(IBM, Riccardo)$	La IBM dà lavoro a Riccardo?
$Dipendente(x, Riccardo)$	Per chi lavora Riccardo?
$Dipendente(IBM, y)$	A chi dà lavoro la IBM?
$Dipendente(x, y)$	Chi dà lavoro a chi?

Queste query formano un **reticolo di sussunzione**, mostrato nella Figura 9.2(a). Il reticolo ha alcune caratteristiche interessanti: il figlio di ogni nodo si ottiene dal padre con una sola sostituzione e il “più alto” discendente comune di due nodi qualsiasi è il risultato dell’applicazione del loro unificatore più generale. Una formula che comprende costanti ripetute ha un reticolo leggermente diverso, mostrato nella Figura 9.2(b). Anche se nella figura non si vedono simboli di funzione, anche questi possono essere incorporati nella struttura a reticolo.

reticolo
di sussunzione

Per prediciati con un numero di argomenti piccolo, si può ottenere un buon compromesso creando un indice per ogni punto nel reticolo di sussunzione: questo richiede un po’ di lavoro in più in fase di memorizzazione, ma velocizza il recupero. Tuttavia, per un prediato con n argomenti, il reticolo corrispondente contiene $O(2^n)$ nodi. Se è consentito l’uso di simboli di funzione, il numero di nodi è anche esponenziale nella dimensione dei termini della formula da memorizzare: questo può condurre a un’esplosione del numero di indici.

Dobbiamo limitare in qualche modo gli indici, utilizzando soltanto quelli che tendono a essere usati più frequentemente nelle query, altrimenti sprecheremo più tempo per creare gli indici di quello che risparmieremo grazie a essi. Potremmo adottare una politica prefissata, come quella di memorizzare gli indici solo per le chiavi composte da un prediato più un singolo argomento, oppure potremmo apprendere una politica adattiva che crea gli indici

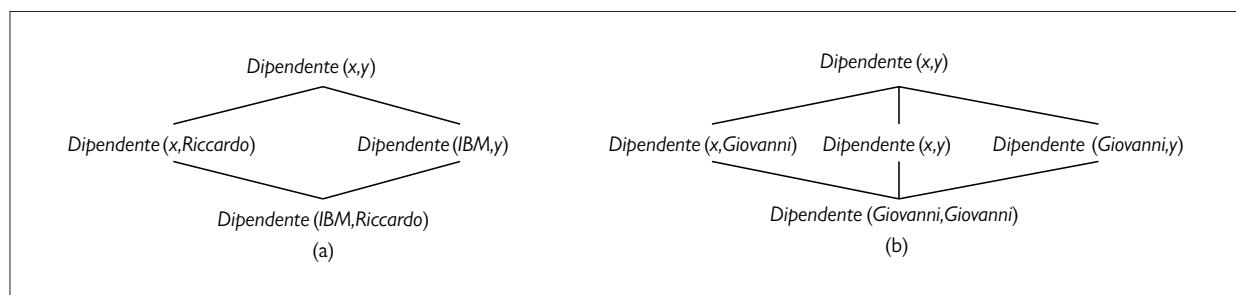


Figura 9.2 (a) Il reticolo di sussunzione il cui nodo più basso è $Dipendente(IBM, Riccardo)$. (b) Il reticolo di sussunzione per la formula $Dipendente(Giovanni, Giovanni)$.

per soddisfare la particolare tipologia di interrogazioni formulate. Per i database commerciali dove il numero di fatti da memorizzare è dell'ordine dei miliardi, il problema è stato oggetto di ampi studi, sviluppo tecnologico e continue ottimizzazioni.

9.3 Concatenazione in avanti

Nel Paragrafo 7.5 del Capitolo 7 abbiamo mostrato un algoritmo di concatenazione in avanti per basi di conoscenza proposizionali composte da clausole definite. Ora estendiamo il concetto alle clausole definite del primo ordine.

Naturalmente esistono formule logiche che non possono essere rappresentate come clausole definite e quindi non sono trattate da questo approccio. Tuttavia, regole della forma *Antecedente* \Rightarrow *Conseguente* sono sufficienti per coprire un'ampia varietà di sistemi del mondo reale.

9.3.1 Clausole definite del primo ordine

Le clausole definite del primo ordine sono disgiunzioni di letterali *esattamente uno dei quali è positivo*. Questo significa che una clausola definita è una formula atomica, oppure è una implicazione il cui antecedente è una congiunzione di letterali positivi e il cui conseguente è un singolo letterale positivo. I quantificatori esistenziali non sono ammessi e quelli universali sono lasciati impliciti: se vedete una x in una clausola definita, significa che c'è un quantificatore隐含的 $\forall x$. Una tipica clausola definita del primo ordine è la seguente:

$$Re(x) \wedge Avido(x) \Rightarrow Malvagio(x)$$

ma anche i letterali $Re(Giovanni)$ e $Avido(y)$ sono clausole definite. I letterali del primo ordine possono includere variabili, perciò $Avido(x)$ è interpretato come “tutti sono avidi” (il quantificatore universale è implicito).

Ora vediamo come si utilizzano le clausole definite per rappresentare il seguente problema:

La legge americana afferma che per un americano è un crimine vendere armi a una nazione ostile. Lo stato di Nono, un nemico dell'America, possiede dei missili, e gli sono stati venduti tutti dal Colonnello West, un americano.

Prima di tutto dobbiamo rappresentare i fatti sotto forma di clausole definite della logica del primo ordine.

“Per un americano è un crimine vendere armi a nazioni ostili”:

$$Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Ostile(z) \Rightarrow Criminale(x). \quad (9.3)$$

“Nono... possiede dei missili”. La formula $\exists x Possiede(Nono, x) \wedge Missile(x)$ si può trasformare in due clausole definite mediante istanziazione esistenziale, introducendo una nuova costante M_1 :

$$Possiede(Nono, M_1) \quad (9.4)$$

$$Missile(M_1). \quad (9.5)$$

“Tutti i missili gli sono stati venduti dal Colonnello West”:

$$Missile(x) \wedge Possiede(Nono, x) \Rightarrow Vende(West, x, Nono). \quad (9.6)$$

Dobbiamo anche sapere che i missili sono armi:

$$Missile(x) \Rightarrow Arma(x). \quad (9.7)$$

E anche che un nemico dell'America viene considerato ostile:

$$Nemico(x, America) \Rightarrow Ostile(x). \quad (9.8)$$

```

function FOL-CA-ASK(KB,  $\alpha$ ) returns una sostituzione oppure false
  inputs: KB, la base di conoscenza, un insieme di clausole definite del primo ordine
   $\alpha$ , la query, una formula atomica

  while true do
    new  $\leftarrow \{ \}$  // L'insieme delle nuove formule inferite a ogni iterazione
    for each regola in KB do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZZA-VARIABILI}(\text{regola})$ 
      for each  $\theta$  tale che  $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$ 
        per qualche  $p'_1, \dots, p'_n$  nella KB
         $q' \leftarrow \text{SUBST}(\theta, q)$ 
        if  $q'$  non si unifica con una formula già presente nella KB o in new then
          aggiungi  $q'$  a new
           $\varphi \leftarrow \text{UNIFY}(q', \alpha)$ 
          if  $\varphi$  non è fallimento then return  $\varphi$ 
        if new =  $\{ \}$  then return false
        aggiungi new alla KB
  
```

Figura 9.3 Un algoritmo di concatenazione in avanti intuitivo, ma inefficiente. A ogni iterazione l'algoritmo aggiunge alla *KB* tutte le formule atomiche che possono essere inferite in un passo dalle formule di implicazione e dalle formule atomiche già presenti nella *KB*. La funzione STANDARDIZZA-VARIABILI sostituisce tutte le variabili presenti nel suo argomento con nuove variabili non ancora utilizzate.

“West è americano”:

$$\text{Americano}(\text{West}). \quad (9.9)$$

“Lo stato di Nono è un nemico dell’America”:

$$\text{Nemico}(\text{Nono}, \text{America}). \quad (9.10)$$

Questa base di conoscenza è di tipo **Datalog**, un linguaggio costituito da clausole definite del primo ordine senza simboli di funzione. Il nome si deve al fatto che il linguaggio è in grado di rappresentare le asserzioni tipicamente contenute nei database relazionali. L’assenza di funzioni rende l’inferenza molto più facile.

Datalog

9.3.2 Un semplice algoritmo di concatenazione in avanti

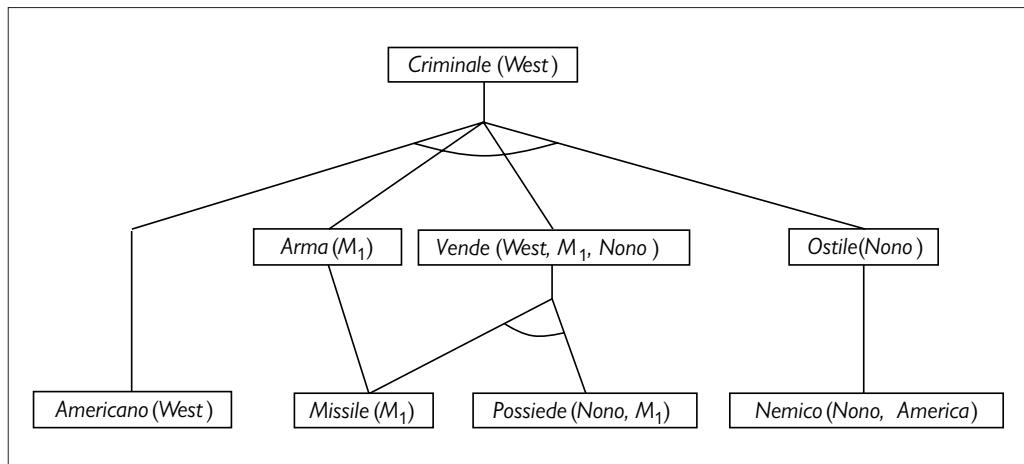
Il primo algoritmo di concatenazione in avanti che consideriamo è semplice, come si vede nella Figura 9.3: partendo dai fatti noti, si fanno scattare tutte le regole le cui premesse sono soddisfatte, aggiungendo le relative conclusioni ai fatti noti. Il processo si ripete finché si trova una risposta (supponendo che ne basti una), oppure non è più possibile aggiungere nuovi fatti. Notate che un fatto non è “nuovo” se consiste solo nella ridenominazione di uno noto. Una formula è la **ridenominazione** di un’altra se tra le due cambiano solo i nomi delle variabili. Per esempio, *Gradisce(x, Gelato)* e *Gradisce(y, Gelato)* sono una la ridenominazione dell’altra; il loro significato è lo stesso: “a tutti piace il gelato”.

ridenominazione

Per illustrare il funzionamento di FOL-CA-ASK useremo il nostro esempio del colonnello criminale. Le formule di implicazione disponibili per la concatenazione sono (9.3), (9.6), (9.7) e (9.8).

Figura 9.4

L'albero di dimostrazione generato dalla concatenazione in avanti sull'esempio del colonnello criminale. I fatti iniziali si trovano al livello più basso, quelli inferiti nella prima iterazione al livello intermedio, quelli inferiti nella seconda iterazione al livello più alto.



Sono necessarie due iterazioni.

- Nella prima iterazione, la regola (9.3) ha delle premesse non soddisfatte. La regola (9.6) è soddisfatta con $\{x/M_1\}$, e si può aggiungere alla base di conoscenza il fatto $Vende(West, M_1, Nono)$. La regola (9.7) è soddisfatta con $\{x/M_1\}$, e si aggiunge $Arma(M_1)$. La regola (9.8) è soddisfatta con $\{x/Nono\}$, e si aggiunge $Ostile(Nono)$.
- Nella seconda iterazione, la regola (9.3) è soddisfatta con $\{x/West, y/M_1, z/Nono\}$, e si può aggiungere $Criminale(West)$.

La Figura 9.4 riporta l'albero di dimostrazione generato. Notate che a questo punto non è più possibile effettuare alcuna nuova inferenza, perché ogni formula che si potrebbe derivare mediante la concatenazione in avanti è già esplicitamente presente nella KB. Una base di conoscenza in questa situazione prende il nome di **punto fisso** del processo inferenziale. I punti fissi raggiunti dalla concatenazione in avanti con clausole definite del primo ordine sono simili a quelli del caso proposizionale, che abbiamo incontrato nel Paragrafo 7.5.4; la differenza principale sta nel fatto che un punto fisso del primo ordine può includere formule atomiche universalmente quantificate.

FOL-CA-ASK è facile da analizzare: per prima cosa è **corretto**, perché ogni inferenza è un'applicazione del Modus Ponens generalizzato. Inoltre è **completo** per le basi di conoscenza di clausole definite; ovvero, è in grado di rispondere a ogni interrogazione le cui soluzioni sono conseguenza logica di una KB composta da sole clausole definite.

Nel caso di basi di conoscenza Datalog, che non contengono simboli di funzione, la dimostrazione della completezza è abbastanza facile. Cominciamo con il contare il numero di fatti possibili che possono essere aggiunti, che determina il numero massimo di iterazioni. Sia k la massima **arità** (numero di argomenti) tra tutti i predicati, p il numero dei predicati ed n quello dei simboli di costante. Palesemente non possono esserci più di pn^k fatti ground distinti, ragion per cui dopo quel numero di iterazioni l'algoritmo deve aver raggiunto un punto fisso. A questo punto possiamo dimostrare la completezza in modo molto simile alla concatenazione in avanti proposizionale (cfr. Paragrafo 7.5.4). I dettagli sul passaggio dalla completezza proposizionale a quella del primo ordine sono forniti per l'algoritmo di risoluzione nel Paragrafo 9.5.

Se le clausole definite includono simboli di funzione, FOL-CA-ASK può generare un numero infinito di fatti, per cui dobbiamo stare attenti. Nel caso in cui la risposta alla query q sia conseguenza logica, possiamo ricorrere al teorema di Herbrand (cfr. Paragrafo 9.1.1) per asserire che l'algoritmo terminerà con successo (cfr. il Paragrafo 9.5 per una discussione

applicata al caso della risoluzione). Se la query non ha risposta, tuttavia, l'algoritmo potrebbe non terminare mai. Per esempio, se la base di conoscenza contiene gli assiomi di Peano,

$$\begin{aligned} & \text{NumNat}(0) \\ & \forall n \quad \text{NumNat}(n) \Rightarrow \text{NumNat}(S(n)), \end{aligned}$$

la concatenazione in avanti aggiungerà $\text{NumNat}(S(0))$, $\text{NumNat}(S(S(0)))$, $\text{NumNat}(S(S(S(0))))$ e così via. Questo non si può evitare in alcun modo: come nel caso generale della logica del primo ordine, anche restringendo il campo alle sole clausole definite il problema della conseguenza logica rimane semidecidibile.

9.3.3 Concatenazione in avanti efficiente

L'algoritmo di concatenazione in avanti riportato nella Figura 9.3 è progettato pensando alla facilità di comprensione e non all'efficienza. Esistono tre fonti di inefficienza: prima di tutto, il ciclo più interno dell'algoritmo cerca di far corrispondere a ogni regola ogni fatto nella base di conoscenza. In secondo luogo, a ogni iterazione l'algoritmo ricontrolla tutte le regole anche se poche aggiunte sono state fatte alla base di conoscenza. In terzo luogo, l'algoritmo può generare molti fatti irrilevanti per il suo obiettivo. Tratteremo uno per volta questi tre aspetti.

Corrispondenze tra regole e fatti conosciuti

Trovare le corrispondenze tra la premessa di una regola e i fatti presenti nella base di conoscenza potrebbe sembrare un problema alquanto semplice. Supponiamo per esempio di voler applicare la regola:

$$\text{Missile}(x) \Rightarrow \text{Arma}(x).$$

Tutto quello che dobbiamo fare è trovare tutti i fatti che unificano con $\text{Missile}(x)$; in una base di conoscenza opportunamente indicizzata questo può essere fatto in un tempo costante per ogni fatto. Ora considerate un regola come:

$$\text{Missile}(x) \wedge \text{Possiede}(\text{Nono}, x) \Rightarrow \text{Vende}(\text{West}, x, \text{Nono}).$$

Ancora una volta possiamo trovare tutti gli oggetti posseduti da Nono in un tempo costante per ogni oggetto; quindi per ognuno di essi potremo verificare se è un missile. Tuttavia, se la base di conoscenza contiene molti oggetti posseduti da Nono e pochi missili, sarebbe più efficiente trovare innanzitutto questi ultimi e dopo controllare se appartengono a Nono. Questo è il problema dell'**ordinamento dei congiunti**: trovare un ordinamento dei congiunti nella premessa di una regola tale che il costo sia minimizzato. Purtroppo trovare l'ordinamento ottimo è un problema NP-difficile, ma sono disponibili buone euristiche: per esempio quella **MRV** (*Minimum Remaining Values*), che abbiamo applicato ai CSP nel Capitolo 6, nel caso ci fossero più oggetti posseduti da Nono che missili ci suggerirebbe di ordinare i congiunti in modo da cercare innanzitutto i missili.

ordinamento
dei congiunti

In effetti, la relazione tra questo **pattern matching** (cioè trovare gli unificatori che unificano la premessa di una regola con i fatti presenti nella base di conoscenza) e il soddisfacimento dei vincoli è molto stretta. Possiamo considerare ogni congiunto come un vincolo sulle variabili contenute: per esempio, $\text{Missile}(x)$ è un vincolo unario su x . Estendendo quest'idea, *possiamo esprimere ogni CSP con dominî finiti con una singola clausola definita e alcuni fatti ground a essa associati*. Considerate il problema di colorazione di una mappa della Figura 6.1, che abbiamo riportato nella Figura 9.5(a): la Figura 9.5(b) fornisce una formulazione equivalente sotto forma di una singola clausola definita. È chiaro che la conclusione *Colorabile()* potrà essere inferita solo se il CSP ha una soluzione. Dato che i CSP includono come casi speciali i problemi 3SAT, possiamo concludere che *cercare il matching tra una clausola definita e un insieme di fatti è un problema NP-difficile*.

pattern matching



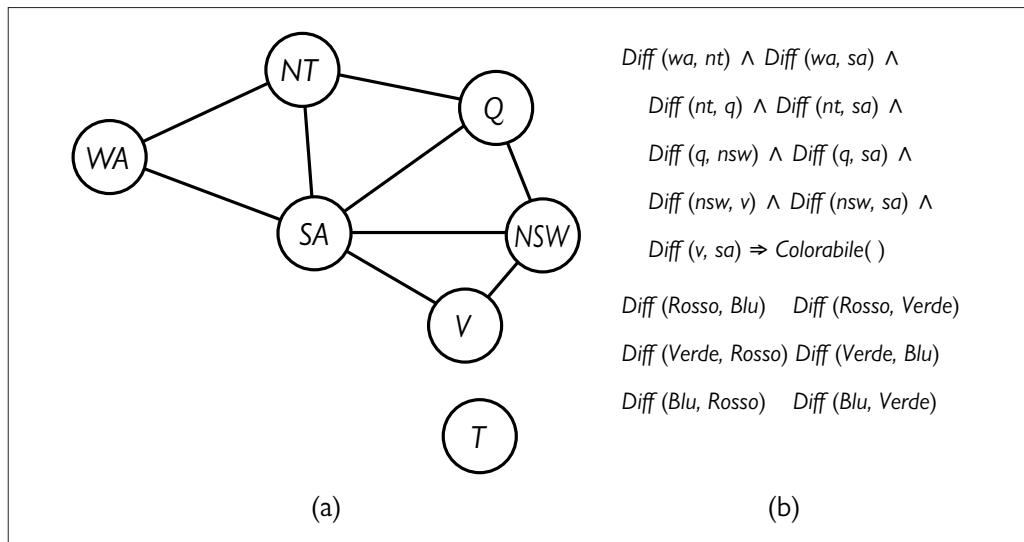


Figura 9.5 (a) Il grafo dei vincoli per la colorazione della mappa dell’Australia. (b) Il CSP della colorazione della mappa espresso con un’unica clausola definita. Ogni area della mappa è rappresentata come variabile il cui valore può essere una delle costanti Rosso, Verde, Blu (le variabili devono avere colori diversi, Diff).

Potrebbe sembrarvi alquanto deprimente che la concatenazione in avanti contenga come ciclo interno un problema NP-difficile. Ci sono tre modi di rallegrarsi.

complessità
relativa ai dati

- Possiamo ricordarci che la maggior parte delle regole, nelle basi di conoscenza reali, sono piccole e semplici (come quelle del nostro esempio del colonnello criminale) e non grandi e complesse (come la formulazione di CSP della Figura 9.5). Nel mondo dei database spesso si dà per scontato che la dimensione delle regole e l’arità dei predicati siano entrambe limitate da una costante e ci si preoccupa solo della **complessità relativa ai dati**, ovvero della complessità dell’inferenza in funzione del numero di fatti ground nella base di conoscenza. È facile dimostrare che la complessità relativa ai dati, per la concatenazione in avanti, è solo polinomiale, non esponenziale.
- Possiamo considerare delle sottoclassi di regole la cui struttura faciliti il matching. Essenzialmente si può dire che ogni clausola Datalog definisce un CSP, ragion per cui il matching sarà trattabile quando lo sarà il CSP corrispondente. Nel Capitolo 6 abbiamo descritto diverse famiglie di CSP trattabili: per esempio se il grafo dei vincoli (quello i cui nodi rappresentano variabili e i cui archi rappresentano vincoli) si riduce a un albero, il CSP può essere risolto in un tempo lineare. Lo stesso identico risultato vale per il matching delle regole; se per esempio eliminiamo SA dalla mappa della Figura 9.5, la clausola risultante è:

$$\text{Diff}(wa, nt) \wedge \text{Diff}(nt, q) \wedge \text{Diff}(q, nsw) \wedge \text{Diff}(nsw, v) \Rightarrow \text{Colorabile}()$$

Che corrisponde al CSP ridotto della Figura 6.12. Gli algoritmi di risoluzione di CSP strutturati ad albero possono essere applicati direttamente al matching delle regole.

- Possiamo cercare di eliminare tentativi di matching ridondanti nell’algoritmo di concatenazione in avanti, come spieghiamo nel seguito.

Concatenazione in avanti incrementale

Quando abbiamo mostrato il funzionamento della concatenazione in avanti sull’esempio del colonnello criminale, abbiamo barato; in particolare, abbiamo omesso una parte del mat-

ching effettuato dall'algoritmo della Figura 9.3. Nella seconda iterazione, per esempio, la regola:

$$\text{Missile}(x) \Rightarrow \text{Arma}(x)$$

ha ancora una corrispondenza in $\text{Missile}(M_1)$, ma naturalmente la conclusione $\text{Arma}(M_1)$ è già nota e non accade nulla. Questo matching ridondante può essere evitato grazie alla seguente osservazione: *ogni nuovo fatto inferito durante l'iterazione t deve derivare da almeno un fatto nuovo inferito nell'iterazione $t - 1$* . È palese infatti che ogni inferenza che non richiede un fatto nuovo, precedentemente non disponibile, sarà stata già effettuata nell'iterazione precedente.



Quest'osservazione conduce naturalmente alla formulazione di un algoritmo di concatenazione in avanti incrementale ove, nell'iterazione t , saranno controllate solo le regole le cui premesse includono un congiunto p_i che unifica con un fatto p'_i inferito nell'iterazione $t - 1$. Il passo di matching allora obbligherà p_i a corrispondere a p'_i , ma permetterà agli altri congiunti della stessa regola di corrispondere a fatti ottenuti in qualsiasi iterazione precedente. Quest'algoritmo genera a ogni iterazione esattamente gli stessi fatti di quello della Figura 9.3, ma è molto più efficiente.

Con un'indicizzazione adeguata è facile identificare tutte le regole che possono essere attivate da ogni fatto, e in effetti molti sistemi operano con una modalità di “aggiornamento” in cui la concatenazione in avanti scatta in risposta a ogni TELL. Le inferenze si susseguono a cascata lungo tutto l'insieme di regole finché non viene raggiunto un punto fisso, e il processo riparte ogni volta che giunge un fatto nuovo.

Tipicamente, l'aggiunta di un fatto provoca l'attivazione di una piccola parte delle regole nella base di conoscenza. Questo significa che un sacco di lavoro viene sprecato nella costruzione ripetuta di corrispondenze parziali che hanno qualche premessa non soddisfatta. Il nostro esempio del colonnello criminale è un po' troppo piccolo per mostrarlo efficacemente, ma notate comunque la costruzione di una corrispondenza parziale nella prima iterazione tra la regola:

$$\text{Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$$

e il fatto $\text{Americano}(\text{West})$. Questa corrispondenza parziale viene scartata e ricostruita daccapo nella seconda iterazione (nella quale viene soddisfatta con successo). Sarebbe meglio mantenere in memoria le corrispondenze parziali anziché scartarle, completandole gradualmente man mano che giungono nuovi fatti.

L'algoritmo Rete³ è stato il primo a occuparsi seriamente di questo problema. L'algoritmo pre-elabora l'insieme di regole nella base di conoscenza per costruire una rete dataflow in cui ogni nodo è un letterale di una premessa di una regola. I legami delle variabili attraversano la rete e sono filtrati via quando non permettono il match con un letterale. Se due letterali in una regola condividono una variabile, come $\text{Vende}(x, y, z) \wedge \text{Ostile}(z)$ nel nostro esempio, i legami di ogni letterale sono filtrati attraverso un nodo di uguaglianza. Un legame di variabile che raggiunge il nodo di un letterale n -ario come $\text{Vende}(x, y, z)$ potrebbe dover aspettare che le altre variabili siano legate prima che il processo possa continuare. In ogni istante, lo stato della rete di Rete contiene tutte le corrispondenze parziali raggiunte, permettendo di evitare una gran quantità di calcoli ripetuti.

algoritmo Rete

I sistemi a rete come quello descritto, e i molti miglioramenti che vi sono stati apportati, sono stati un componente chiave dei cosiddetti **sistemi di produzioni**, che sono stati tra i pri-

sistemi di produzioni

³ Il nome deriva dalla parola latina *rete*, che ha lo stesso significato di quella italiana.

mi sistemi di concatenazione in avanti di grande diffusione.⁴ Il sistema XCON (originariamente denominato R1, McDermott, 1982) era basato su un sistema di produzioni. XCON conteneva diverse migliaia di regole per la progettazione di configurazioni di componenti per computer per i clienti della Digital Equipment Corporation. Fu uno dei primi chiari successi commerciali nel campo emergente dei sistemi esperti. In seguito sono stati costruiti molti sistemi basati sulla stessa tecnologia, che è stata implementata nel linguaggio di uso generale OPS-5.

architettura cognitiva

I sistemi di produzioni sono popolari anche nel campo delle **architetture cognitive** (che si ripropongono cioè di modellare il ragionamento umano), come ACT (Anderson, 1983) e SOAR (Laird *et al.*, 1987). In tali sistemi la “memoria di lavoro” rappresenta quella umana a breve termine, mentre le produzioni fanno parte della rappresentazione della memoria a lungo termine. A ogni ciclo si cercano corrispondenze tra le produzioni e i fatti contenuti nella memoria di lavoro. Una produzione le cui condizioni sono soddisfatte può aggiungere o cancellare fatti dalla memoria di lavoro. A differenza di quello che accade di solito con i database, i sistemi di produzioni hanno tipicamente molte regole e un numero relativamente piccolo di fatti. Sfruttando tecniche di matching adeguatamente ottimizzate, alcuni sistemi moderni possono operare in tempo reale con più di un milione di regole.

Fatti irrilevanti

Un’altra fonte di inefficienza è data dal fatto che la concatenazione in avanti effettua tutte le inferenze consentite dai fatti conosciuti, *anche se sono irrilevanti per raggiungere l’obiettivo*. Nel nostro esempio del colonnello criminale, non c’erano regole da cui potessero essere tratte conclusioni irrilevanti. Ma se vi fossero molte regole che descrivessero le abitudini alimentari degli americani, o le componenti e i prezzi dei missili, FOL-CA-ASK genererebbe conclusioni irrilevanti.

database deduttivo

Un modo di evitare di trarre conclusioni irrilevanti è usare la concatenazione all’indietro, che descriveremo nel Paragrafo 9.4. Un altro modo è restringere la concatenazione in avanti a un insieme selezionato di regole, come in LP-CA-CONSEGUE? (Figura 7.15). Un terzo metodo è stato sviluppato dalla comunità dei **database deduttivi**, che sono database come quelli relazionali, ma che utilizzano come strumento di inferenza standard la concatenazione in avanti anziché le query SQL. L’idea è riscrivere l’insieme di regole utilizzando informazione presa dall’obiettivo in modo tale che solo i legami rilevanti – quelli che appartengono al cosiddetto “insieme magico” o **magic set** – siano presi in considerazione durante l’inferenza in avanti. Per esempio, se l’obiettivo è *Criminale(West)*, la regola che ha come conclusione *Criminale(x)* sarà riscritta in modo da includere un congiunto addizionale che vincola il valore di *x*:

$$\text{Magic}(x) \wedge \text{Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x).$$

magic set

Naturalmente, alla KB bisognerà aggiungere il fatto *Magic(West)*. In questo modo, anche se la base di conoscenza contiene fatti che riguardano milioni di americani, durante il processo di inferenza in avanti sarà considerato il solo colonnello West. Il processo completo per definire gli insiemi magici e riscrivere la base di conoscenza è troppo complesso per approfondirlo qui ma l’idea, a grandi linee, consiste nell’eseguire una specie di inferenza all’indietro “generica” partendo dall’obiettivo, per determinare quali legami di variabili devono essere vincolati. La soluzione dei magic set può quindi essere considerata una specie di approccio ibrido che sfrutta l’inferenza in avanti unita a una fase di pre-elaborazione all’indietro.

⁴ Nei **sistemi di produzioni**, la parola **produzione** indica una regola condizione-azione.

```

function FOL-CI-ASK(KB, query) returns un generatore di sostituzioni
  return FOL-CI-OR(KB, query, { })

function FOL-CI-OR(KB, obiettivo, θ) returns una sostituzione
  for each regola in ESTRAI-REGOLE-PER-OBIETTIVO(KB, obiettivo) do
    (lhs, rhs)  $\leftarrow$  STANDARDIZZA-VARIABILI(regola)
    for each  $\theta'$  in FOL-CI-AND(KB, lhs, UNIFY(rhs, obiettivo, θ)) do
      yield  $\theta'$ 

function FOL-CI-AND(KB, obiettivi, θ) returns una sostituzione
  if  $\theta = \text{fallimento}$  then return
  else if LUNGHEZZA(obiettivi) = 0 then yield  $\theta$ 
  else
    primo, resto  $\leftarrow$  PRIMO(obiettivi), RESTO(obiettivi)
    for each  $\theta'$  in FOL-CI-OR(KB, SUBST(θ, primo), θ) do
      for each  $\theta''$  in FOL-CI-AND(KB, resto, θ') do
        yield  $\theta''$ 

```

Figura 9.6 Un semplice algoritmo di concatenazione all'indietro per basi di conoscenza del primo ordine.

9.4 Concatenazione all'indietro

La seconda grande famiglia di algoritmi di inferenza logica utilizza la **concatenazione all'indietro** su clausole definite. Questi algoritmi procedono dall'obiettivo seguendo le regole a ritroso per trovare fatti noti che supportino la dimostrazione.

9.4.1 Un algoritmo di concatenazione all'indietro

La Figura 9.6 mostra un semplice algoritmo di concatenazione all'indietro per clausole definite. FOL-CI-ASK(*KB, obiettivo*) sarà dimostrata se la base di conoscenza contiene una regola della forma *lsh* \Rightarrow *obiettivo*, dove *lsh* (*left-hand side* cioè lato sinistro), è una lista di congiunti. Un fatto atomico come *Americano(West)* è considerato come una clausola in cui *lsh* è la lista vuota. Ora una query che contiene variabili può essere dimostrata in più modi. Per esempio, la query *Persona(x)* potrebbe essere dimostrata con la sostituzione $\{x/\text{Giovanni}\}$ o anche con $\{x/\text{Riccardo}\}$. Perciò implementiamo FOL-CI-ASK come generatore, una funzione che restituisce il controllo più volte, fornendo ogni volta un possibile risultato (cfr. Appendice B).

La concatenazione all'indietro è un tipo particolare di ricerca AND/OR, dove la parte OR si deve al fatto che la query obiettivo può essere dimostrata da qualsiasi regola della base di conoscenza e la parte AND si deve al fatto che tutti i congiunti della lista *lsh* di una clausola devono essere dimostrati. FOL-CI-OR opera cercando tutte le clausole che potrebbero unificare con l'obiettivo, standardizzando le variabili nella clausola in modo che risultino variabili del tutto nuove e poi, se la *rhs* (cioè la parte destra) della clausola unifica con l'obiettivo, dimostrando ogni congiunto della lista *lsh*, utilizzando FOL-CI-AND. Tale funzione opera dimostrando i congiunti uno per uno e tenendo traccia della sostituzione accumulata mentre procede. La Figura 9.7 mostra l'albero di dimostrazione per derivare *Criminale(West)* dalle formule da (9.3) a (9.10).

La concatenazione all'indietro, così come l'abbiamo scritta, è chiaramente un algoritmo di ricerca in profondità: questo significa che i suoi requisiti spaziali sono lineari con le di-

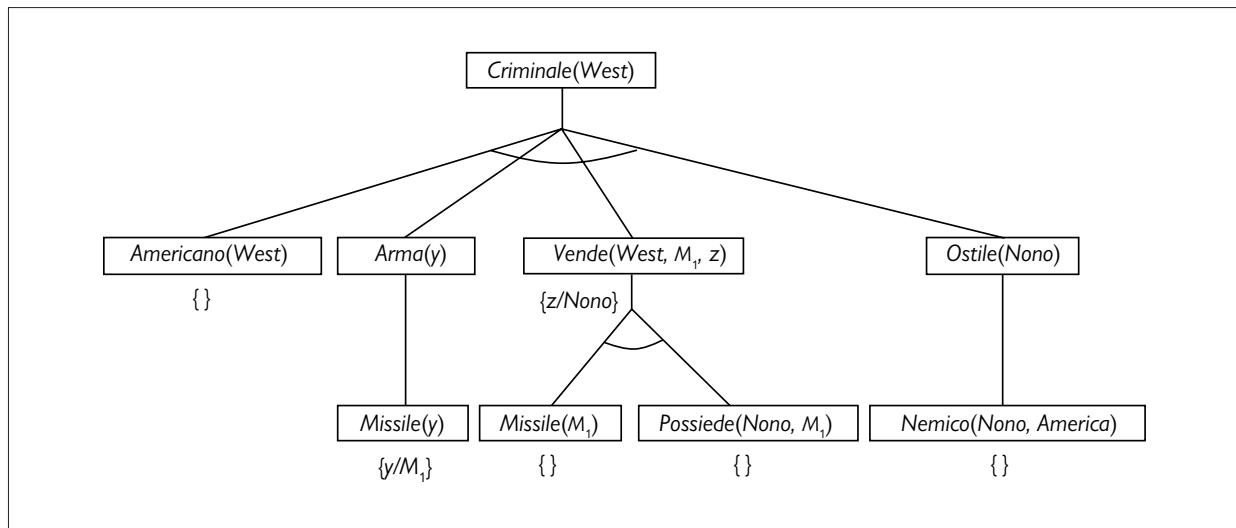


Figura 9.7 L'albero costruito con la concatenazione all'indietro per dimostrare che il Colonnello West è un criminale. L'albero va letto in profondità e da sinistra a destra. Per provare *Criminale(West)* dobbiamo dimostrare i quattro congiunti al livello appena inferiore. Alcuni di essi sono nella base di conoscenza, altri richiedono un'ulteriore concatenazione all'indietro. I legami per ogni unificazione che ha successo sono indicati accanto al sotto-obiettivo corrispondente. Notate che una volta che un sotto-obiettivo in una congiunzione ha successo, la sua sostituzione viene applicata anche ai sotto-obiettivi successivi. In questo modo quando FOL-CI-ASK arriva a considerare l'ultimo congiunto, che originariamente era *Ostile(z)*, *z* è già legata a *Nono*.

mensioni della dimostrazione. Questo significa anche che, a differenza di quella in avanti, la concatenazione all'indietro soffre del problema degli stati ripetuti e dell'incompletezza. Nonostante queste limitazioni, la concatenazione all'indietro si è dimostrata efficace ed è ampiamente utilizzata nei linguaggi di programmazione logica.

9.4.2 Programmazione logica

La programmazione logica è uno strumento che arriva vicino ad avverare l'ideale dichiarativo che abbiamo illustrato nel Capitolo 7: la possibilità di costruire sistemi esprimendo semplicemente la conoscenza in un linguaggio formale e risolvendo i problemi attraverso un processo di inferenza applicato a tale conoscenza. L'ideale è riassunto dall'equazione di Robert Kowalski:

$$\text{Algoritmi} = \text{Logica} + \text{Controllo}.$$

Prolog

Il linguaggio di programmazione logica più usato è di gran lunga **Prolog**. La sua applicazione è normalmente rivolta allo sviluppo rapido di prototipi e alle attività che richiedono manipolazione di simboli, come la scrittura di compilatori (Van Roy, 1990), ed è anche utilizzato per il riconoscimento sintattico (*parsing*) del linguaggio naturale (Pereira e Warren, 1980). Sono stati scritti molti sistemi esperti in Prolog nei domini legale, medico, finanziario e molti altri.

I programmi Prolog consistono in un insieme di clausole definite espresse con una notazione un po' differente da quella standard della logica del primo ordine. Le variabili sono scritte in maiuscolo, le costanti in minuscolo, l'opposto della nostra convenzione per la logica. Per separare i congiunti nelle clausole si utilizzano le virgolette, e la clausola è scritta “all'indietro” rispetto a come siamo abituati: invece di $A \wedge B \Rightarrow C$, in Prolog abbiamo $C :- A, B$. Ecco un tipico esempio:

```
criminale(X) :- americano(X), arma(Y), vende(X, Y, Z), ostile(Z).
```

In Prolog la notazione $[E|L]$ denota una lista il cui primo elemento è E e il resto è L . Ecco un programma Prolog `concatena(X, Y, Z)`, che ha successo se la lista Z è il risultato della concatenazione delle liste X e Y :

```
concatena([], Y, Y).
concatena([A|X], Y, [A|Z]) :- concatena(X, Y, Z).
```

Possiamo leggere queste clausole come: (1) concatenare la lista vuota e la lista Y produce la stessa lista Y ; (2) $[A|Z]$ è il risultato della concatenazione di $[A|X]$ e Y , a patto che Z sia il risultato della concatenazione di X e Y . Nella maggior parte dei linguaggi di alto livello si può scrivere una funzione ricorsiva simile che descrive come concatenare due liste, ma la definizione del Prolog è molto più potente, perché descrive una *relazione* che vale fra i tre argomenti, anziché una *funzione* calcolata da due argomenti. Se per esempio scriviamo la query `concatena(X, Y, [1, 2, 3])` – quali due liste possono essere concatenate per dare la lista $[1, 2, 3]$? In Prolog otteniamo le soluzioni:

```
X= []          Y= [1,2,3];
X= [1]         Y= [2,3];
X= [1,2]        Y= [3];
X= [1,2,3]      Y= []
```

L'esecuzione dei programmi Prolog è svolta mediante una concatenazione all'indietro in profondità, in cui le clausole vengono provate nell'ordine con cui sono state scritte nella base di conoscenza. Il modo in cui è stato progettato il Prolog rappresenta un compromesso tra l'obiettivo di creare un linguaggio dichiarativo e quello di ottenere efficienza in esecuzione. Alcuni aspetti del Prolog differiscono dell'inferenza logica standard.

- Il Prolog utilizza la semantica dei database del Paragrafo 8.2.8 anziché la semantica del primo ordine, e questo si vede nel modo in cui tratta uguaglianza e negazione (cfr. Paragrafo 9.4.4).
- È presente un insieme di funzioni aritmetiche predefinite. I letterali che usano questi simboli di funzione sono “dimostrati” eseguendo codice e non mediante ulteriori inferenze. Per esempio, l'obiettivo “ $X \text{ is } 4+3$ ” risulta vero con X legata a 7. D'altra parte, l'obiettivo “ $5 \text{ is } X+Y$ ” fallisce sempre, perché le funzioni aritmetiche predefinite non risolvono equazioni arbitrarie.
- Alcuni predicati predefiniti hanno effetti collaterali. Tra questi ci sono quelli di input-output e quelli di `assert/retract` utilizzati per modificare la base di conoscenza. Predicati simili non hanno alcuna controparte nella logica e possono produrre risultati confusi: per esempio, se si asseriscono alcuni fatti in un ramo dell'albero di dimostrazione che a un certo punto fallisce.
- L'algoritmo di unificazione del Prolog non include il **controllo di occorrenza**. Questo significa che potranno essere effettuate inferenze scorrette; nella pratica questo non rappresenta quasi mai un problema.
- Il Prolog utilizza la ricerca con concatenazione all'indietro in profondità senza controlli sulla ricorsione infinita. Per questo è un linguaggio di programmazione facile da usare e molto veloce quando è utilizzato in modo appropriato, ma alcuni programmi che sembrano validi come logica in effetti non terminano mai.

9.4.3 Inferenza ridondante e cicli infiniti

Passiamo ora a occuparci del tallone d'Achille del Prolog: l'incompatibilità tra la ricerca in profondità e gli alberi di ricerca che presentano stati ripetuti e cammini infiniti. Considerate il seguente programma logico che decide se esiste un cammino tra due punti di un grafo orientato:

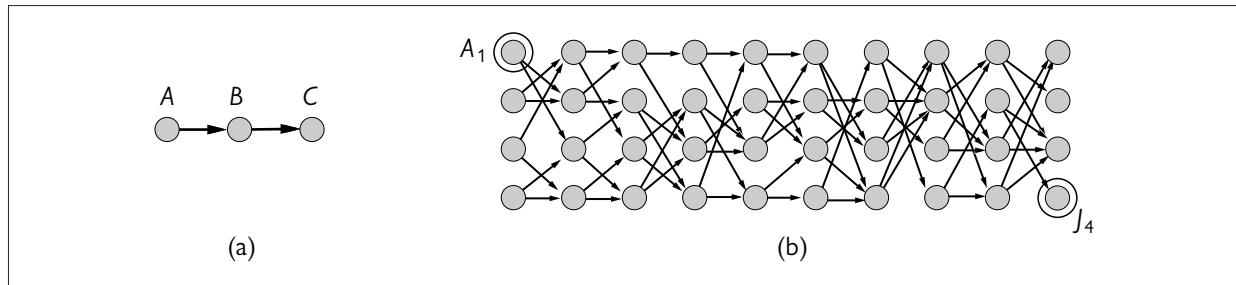


Figura 9.8 (a) Trovare un cammino da A a C può portare il Prolog in un ciclo infinito. (b) Un grafo in cui ogni nodo è collegato a due successori casuali del livello successivo. Trovare un cammino da A_1 a J_4 richiede 877 inferenze.

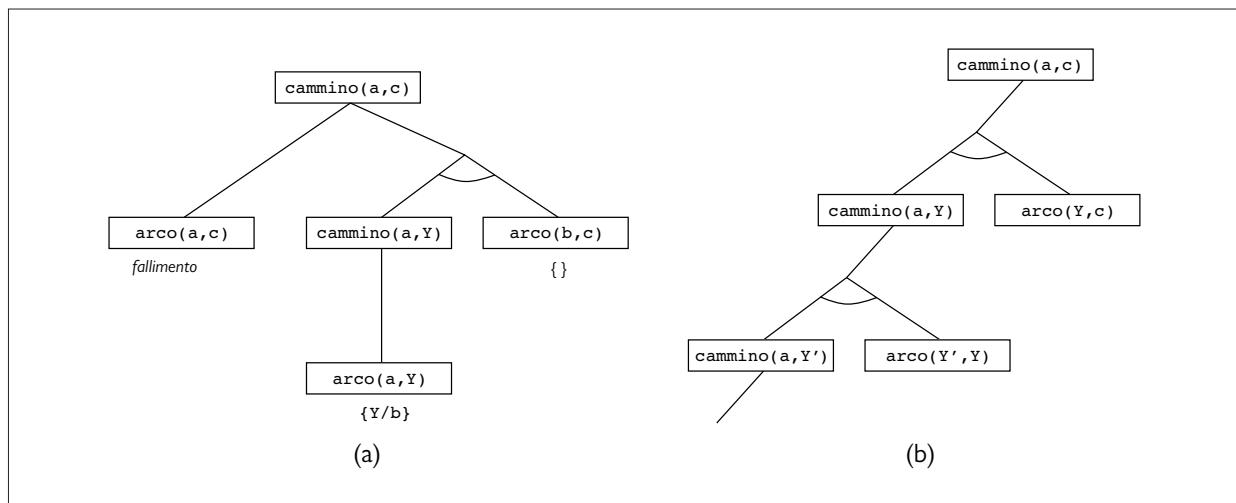


Figura 9.9 (a) Dimostrazione che esiste un cammino da A a C . (b) Quando le clausole sono in ordine “sbagliato”, può essere generato un albero di dimostrazione infinito.

```
cammino(X,Z) :- arco(X,Z).
cammino(X,Z) :- cammino(X,Y), arco(Y,Z).
```

La Figura 9.8(a) mostra un semplice grafo a tre nodi, descritto dai fatti `arco(a,b)` e `arco(b,c)`. Con il programma qui sopra, la query `cammino(a,c)` genera l’albero di dimostrazione riportato nella Figura 9.9(a). D’altra parte, se scriviamo le due clausole in ordine inverso

```
cammino(X,Z) :- cammino(X,Y), arco(Y,Z).
cammino(X,Z) :- arco(X,Z).
```

Prolog segue il cammino infinito mostrato nella Figura 9.9(b). Ne consegue che Prolog è **incompleto** come dimostratore di teoremi per clausole definite; l’esempio mostra che questo avviene anche nel caso di programmi Datalog. La ragione è che, per certe basi di conoscenza, Prolog non riesce a dimostrare formule che sono effettivamente conseguenze logiche. Notate che la concatenazione in avanti non ha questo problema: una volta inferite `cammino(a,b)`, `cammino(b,c)` e `cammino(a,c)`, la concatenazione in avanti si arresta.

La concatenazione all’indietro in profondità ha anche il problema di eseguire molti calcoli ridondanti. Per esempio, per trovare il cammino da A_1 a J_4 nella Figura 9.8(b), Prolog esegue

877 inferenze, la maggior parte delle quali per cercare tutti i cammini possibili verso nodi da cui l'obiettivo è irraggiungibile. Questo problema è simile a quello degli stati ripetuti che abbiamo discusso nel Capitolo 3. Il numero totale di inferenze può crescere esponenzialmente con il numero di fatti ground generati. Applicando la concatenazione in avanti, invece, saranno generati al più n^2 fatti cammino (X, Y) che collegano n nodi. Per il problema della Figura 9.8(b), sono necessarie solo 62 inferenze.

La concatenazione in avanti applicata a problemi di ricerca su grafo è un esempio di **programmazione dinamica**, in cui le soluzioni ai sottoproblemi sono costruite incrementalmente partendo da quelle dei sottoproblemi più piccoli, e memorizzate in una cache per evitare di ricalcolarle. Possiamo ottenere un effetto simile in un sistema che utilizza la concatenazione all'indietro, con la differenza che in questo caso suddividiamo obiettivi grandi in altri più piccoli, anziché costruire obiettivi più grandi a partire dai più piccoli.

In ogni caso, la chiave sta nel memorizzare i risultati intermedi in modo da evitare le duplicazioni. Questo è l'approccio adottato dai sistemi di **programmazione logica con tabelle**, che sfruttano meccanismi efficienti di salvataggio e recupero dati. La programmazione logica con tabelle è guidata dall'obiettivo come la concatenazione all'indietro, ma è efficiente come quella in avanti; inoltre è completa per basi di conoscenza Datalog, il che significa che il programmatore deve preoccuparsi meno dei cicli infiniti (rimane possibile ottenere un ciclo infinito con predicati quali $\text{padre}(X, Y)$ che fanno riferimento a un numero di oggetti potenzialmente illimitato).

programmazione dinamica

programmazione logica con tabelle

9.4.4 Semantica dei database di Prolog

Prolog utilizza la semantica dei database, descritta nel Paragrafo 8.2.8. L'ipotesi dei nomi unici afferma che ogni costante Prolog e ogni termine ground si riferisce a un oggetto distinto, e l'ipotesi del mondo chiuso afferma che le sole formule vere sono quelle che sono conseguenze logiche della base di conoscenza. Non è possibile asserire che una formula è falsa, in Prolog. Ciò rende questo linguaggio meno espressivo rispetto alla logica del primo ordine, ma contribuisce a renderlo più efficiente e conciso. Considerate le seguenti asserzioni su alcuni corsi di studio:

$$\text{Corso}(CS, 101), \text{Corso}(CS, 102), \text{Corso}(CS, 106), \text{Corso}(EE, 101). \quad (9.11)$$

Sotto l'ipotesi dei nomi unici, CS ed EE sono diversi (come 101, 102 e 106), perciò ci sono quattro corsi distinti. Sotto l'ipotesi del mondo chiuso non ci sono altri corsi, perciò i corsi sono esattamente quattro. Ma se queste fossero asserzioni in FOL anziché nella semantica dei database, potremmo soltanto dire che ci sono da uno a infiniti corsi. Questo perché le asserzioni (in FOL) non negano la possibilità che siano offerti anche altri corsi non citati, né affermano che i corsi citati siano diversi l'uno dall'altro. Se volessimo tradurre la formula (9.11) in FOL, otterremmo la formula seguente:

$$\begin{aligned} \text{Corso}(d, n) \Leftrightarrow & (d = CS \wedge n = 101) \vee (d = CS \wedge n = 102) \\ & \vee (d = CS \wedge n = 106) \vee (d = EE \wedge n = 101). \end{aligned} \quad (9.12)$$

Questo si dice **completamento** della formula (9.11) ed esprime in FOL il concetto che vi sono al massimo quattro corsi. Per esprimere in FOL il concetto che vi sono almeno quattro corsi, dobbiamo scrivere il completamento del predicato di uguaglianza:

$$\begin{aligned} x = y \Leftrightarrow & (x = CS \wedge y = CS) \vee (x = EE \wedge y = EE) \vee (x = 101 \wedge y = 101) \\ & \vee (x = 102 \wedge y = 102) \vee (x = 106 \wedge y = 106). \end{aligned}$$

completamento

Il completamento è utile per comprendere la semantica dei database, ma per scopi pratici, se il problema può essere descritto con la semantica dei database, è più efficiente ragionare con Prolog o un altro sistema con semantica dei database, anziché tradurre in FOL e ragionare con un dimostratore di teoremi FOL.

9.4.5 Programmazione logica a vincoli

Nella nostra discussione della concatenazione in avanti (Paragrafo 9.3) abbiamo mostrato che i problemi di soddisfacimento di vincoli (CSP) possono essere espressi sotto forma di clausole definite. Il Prolog standard risolve tali problemi esattamente come l'algoritmo con backtracking illustrato nella Figura 6.5.

Dato che il backtracking enumera i dominî delle variabili, è applicabile solo a CSP a **domî finiti**. Usando il gergo del Prolog, diciamo che dev'esserci un numero finito di soluzioni per ogni obiettivo con variabili libere (per esempio, possiamo pensare al problema di colorare una mappa in cui ogni variabile può assumere un colore tra quattro disponibili). I CSP a dominî infiniti, come quelli che prevedono variabili intere o reali, richiedono algoritmi molto diversi, come la propagazione dei vincoli o la programmazione lineare.

Consideriamo il seguente esempio. Definiamo `triangolo(X, Y, Z)` come un predicato che vale se i tre argomenti sono numeri che soddisfano la disuguaglianza triangolare:

```
triangolo(X,Y,Z) :-  
    X>0, Y>0, Z>0, X+Y>Z, Y+Z>X, X+Z>Y.
```

Se sottoponiamo al Prolog la query `triangolo(3, 4, 5)`, la risposta è positiva. Al contrario, l'interrogazione `triangolo(3, 4, Z)` non potrà avere alcuna soluzione, perché il Prolog non può gestire il sotto-obiettivo `Z>0`. Non possiamo confrontare un valore illimitato con lo zero.

La **programmazione logica a vincoli** (o CLP, dall'acronimo inglese) permette di *vincolare* le variabili anziché *legarle*. Una soluzione CLP è il più specifico insieme di vincoli sulle variabili della query che può essere derivato dalla base di conoscenza. Per esempio, la soluzione dell'interrogazione `triangolo(3, 4, Z)` è il vincolo $7 > Z > 1$. I programmi logici standard a questo punto diventano un caso speciale di CLP che comprende solo vincoli di ugualanza, cioè legami.

I sistemi CLP incorporano diversi algoritmi per la risoluzione dei vincoli permessi nel linguaggio. Un sistema che permette la formulazione di disuguaglianze lineari su variabili reali, per esempio, potrebbe includere un algoritmo specializzato di programmazione lineare per la loro soluzione. I sistemi CLP adottano anche un approccio molto più flessibile alla risoluzione delle query standard della programmazione logica: per esempio, invece di applicare sempre (poniamo) un algoritmo di ricerca in profondità con backtracking da sinistra a destra, potrebbero utilizzare uno degli algoritmi più efficienti che abbiamo discusso nel Capitolo 6, come l'ordinamento euristico dei congiunti, il backjumping, il condizionamento con insieme di taglio e così via. Si può dire quindi che uniscono elementi tipici degli algoritmi di soddisfacimento di vincoli, della programmazione logica e delle basi di dati deduttive.

Sono stati definiti molti sistemi che consentono al programmatore di esercitare maggior controllo sull'ordine di ricerca dell'inferenza. Il linguaggio MRS (Genesereth e Smith, 1981; Russell, 1985) permette al programmatore di scrivere **metaregole** per specificare quali congiunti provare per primi. L'utente potrebbe scrivere una regola per far sì che l'obiettivo con il minor numero di variabili sia il primo a essere considerato, o aggiungere regole specifiche del dominio per la gestione di particolari predicati.

9.5 Risoluzione

L'ultima delle nostre tre famiglie di sistemi logici, l'unica che funziona per qualsiasi base di conoscenza e non solo per le clausole definite, è la **risoluzione**. Abbiamo visto nel Paragrafo 7.5 che la risoluzione proposizionale è una procedura di inferenza completa per la logica proposizionale; in questo paragrafo estendiamo la sua applicazione alla logica del primo ordine.

9.5.1 Forma normale congiuntiva per la logica del primo ordine

Il primo passo consiste nel convertire le formule in **forma normale congiuntiva** (CNF, dall'acronimo inglese), ovvero in una congiunzione di clausole, ognuna delle quali è formata da una disgiunzione di letterali.⁵ In CNF i letterali possono contenere variabili, che sono sempre considerate universalmente quantificate. Per esempio, la formula:

$$\forall x \text{ Americano}(x) \wedge \text{Arma}(y) \wedge \text{Vende}(x, y, z) \wedge \text{Ostile}(z) \Rightarrow \text{Criminale}(x)$$

diventa in CNF:

$$\neg \text{Americano}(x) \vee \neg \text{Arma}(y) \vee \neg \text{Vende}(x, y, z) \vee \neg \text{Ostile}(z) \vee \text{Criminale}(x).$$

Il punto chiave è che *ogni formula della logica del primo ordine può essere convertita in una formula CNF inferenzialmente equivalente.*



La procedura per la conversione in CNF è molto simile a quella che abbiamo visto nel Paragrafo 7.5.2. La differenza principale sorge dalla necessità di eliminare i quantificatori esistenziali. Illustriamo la procedura traducendo la formula “chiunque ami tutti gli animali è amato da qualcuno”:

$$\forall x [\forall y \text{ Animale}(y) \Rightarrow \text{Ama}(x, y)] \Rightarrow [\exists y \text{ Ama}(y, x)].$$

I passi sono i seguenti.

- Eliminazione delle implicazioni: sostituiamo $P \Rightarrow Q$ con $\neg P \vee Q$. Per la nostra formula di esempio occorre farlo due volte:

$$\begin{aligned} &\forall x \neg [\forall y \text{ Animale}(y) \Rightarrow \text{Ama}(x, y)] \vee [\exists y \text{ Ama}(y, x)] \\ &\forall x \neg [\forall y \neg \text{Animale}(y) \vee \text{Ama}(x, y)] \vee [\exists y \text{ Ama}(y, x)]. \end{aligned}$$

- **Spostamento all'interno delle negazioni:** oltre alle regole consuete sui connettivi negati, ce ne servono altre per i quantificatori negati. Quindi:

$$\begin{aligned} \neg \forall x p &\quad \text{diventa} \quad \exists x \neg p \\ \neg \exists x p &\quad \text{diventa} \quad \forall x \neg p. \end{aligned}$$

La nostra formula passa attraverso le seguenti trasformazioni:

$$\begin{aligned} &\forall x [\exists y \neg (\neg \text{Animale}(y) \vee \text{Ama}(x, y))] \vee [\exists y \text{ Ama}(y, x)] \\ &\forall x [\exists y \neg \neg \text{Animale}(y) \wedge \neg \text{Ama}(x, y)] \vee [\exists y \text{ Ama}(y, x)] \\ &\forall x [\exists y \text{ Animale}(y) \wedge \neg \text{Ama}(x, y)] \vee [\exists y \text{ Ama}(y, x)]. \end{aligned}$$

Notate come il quantificatore universale ($\forall y$) nella premessa dell'implicazione sia diventato un quantificatore esistenziale. Ora la formula si legge “O esiste un animale che x non ama, oppure (se questo non è il caso) qualcuno ama x ”. È chiaro che il significato originale della formula è stato preservato.

- **Standardizzazione delle variabili:** in formule come $(\forall x P(x)) \vee (\exists x Q(x))$, che usano due volte lo stesso nome di variabile, si deve cambiare il nome di una delle due. Questo ci permetterà di evitare confusione più tardi, quando dovremo eliminare i quantificatori. Risulta così:

$$\forall x [\exists y \text{ Animale}(y) \wedge \neg \text{Ama}(x, y)] \vee [\exists z \text{ Ama}(z, x)].$$

⁵ Una clausola può anche essere rappresentata da un'implicazione formata da una congiunzione di atomi nella premessa e una disgiunzione di atomi nella conclusione (Esercizio 9.DISJ). Prende il nome di **forma normale implicativa** o **forma di Kowalski** (in particolare quando è scritta con un simbolo di implicazione che va da destra a sinistra (Kowalski, 1979b)) ed è generalmente molto più facile da leggere rispetto a una disgiunzione con molti letterali negati.

Skolemizzazione

Skolemizzazione: è il processo di rimozione dei quantificatori esistenziali per eliminazione. Nel caso più semplice si riduce alla regola di istanziazione esistenziale del Paragrafo 9.1: $\exists x P(x)$ si traduce in $P(A)$, dove A è una nuova costante. Tuttavia, non possiamo applicare l'istanziazione esistenziale alla formula precedente, perché non corrisponde allo schema $\exists v \alpha$; soltanto parti della formula corrispondono allo scherma. Se applichiamo questa regola alle due parti corrispondenti, otteniamo:

$$\forall x [Animale(A) \wedge \neg Ama(x, A)] \vee Ama(B, x),$$

che ha un significato totalmente sbagliato, infatti afferma che ogni individuo non ama un particolare animale A , oppure è a sua volta amato da una qualche specifica entità B . In realtà, la formula originale permetteva a ogni individuo di non amare un animale diverso, o di essere amata da una persona diversa. Quindi le entità di Skolem devono dipendere da x :

$$\forall x [Animale(F(x)) \wedge \neg Ama(x, F(x))] \vee Ama(G(x), x).$$

funzioni di Skolem

F e G sono dette **funzioni di Skolem**. Come regola generale, gli argomenti di una funzione di Skolem sono tutte le variabili universalmente quantificate all'interno del cui campo d'azione appare un quantificatore esistenziale. Come nel caso della istanziazione esistenziale, la formula Skolemizzata è soddisfacibile esattamente nei casi in cui lo è quella originale.

- **Omissione dei quantificatori universali:** a questo punto, tutte le variabili restanti devono essere universalmente quantificate. Possiamo quindi omettere i quantificatori senza perdere informazioni:

$$[Animale(F(x)) \wedge \neg Ama(x, F(x))] \vee Ama(G(x), x).$$

- **Distribuzione di \vee su \wedge :**

$$[Animale(F(x)) \vee Ama(G(x), x)] \wedge [\neg Ama(x, F(x)) \vee Ama(G(x), x)].$$

Quest'ultimo passo potrebbe anche richiedere l'appiattimento di congiunzioni e disgiunzioni annidate.

Ora la formula è in CNF ed è composta da due clausole. È molto più difficile da leggere rispetto alla formula originale con implicazioni (un aiuto consisterebbe nello specificare che la funzione di Skolem $F(x)$ si riferisce a un animale potenzialmente non amato da x , mentre $G(x)$ si riferisce a qualcuno che potrebbe amare x). Fortunatamente, gli esseri umani hanno raramente bisogno di esaminare direttamente le formule CNF; la traduzione si può facilmente automatizzare.

9.5.2 La risoluzione come regola di inferenza

La regola di risoluzione per le clausole del primo ordine è semplicemente una versione “sollevata” tramite lifting di quella che abbiamo già presentato nel Paragrafo 7.5.2 per il caso proposizionale. Due clausole, che grazie alla standardizzazione separata presumiamo sempre non avere alcuna variabile in comune, possono essere risolte se contengono letterali complementari. I letterali proposizionali sono complementari se uno è la negazione dell'altro; quelli di primo ordine lo sono se uno *unifica* con la negazione dell'altro. Quindi abbiamo:

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n}{\text{SUBST}(\theta, \ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

dove $\text{UNIFY}(\ell_i, \neg m_j) = \theta$. Per esempio, possiamo risolvere le due clausole:

$$[Animale(F(x)) \vee Ama(G(x), x)] \quad \text{e} \quad [\neg Ama(u, v) \vee \neg Uccide(u, v)]$$

eliminando i letterali complementari $Ama(G(x), x)$ e $\neg Ama(u, v)$, con unificatore $\theta = \{u/G(x), v/x\}$ e producendo la clausola **risolvente**:

$$[Animale(F(x)) \vee \neg Uccide(G(x), x)].$$

Questa regola si chiama **risoluzione binaria**, perché risolve esattamente due letterali. La regola di risoluzione binaria, di per sé, non fornisce una procedura di inferenza completa: per questo si devono risolvere i sottoinsiemi di letterali unificabili di ogni clausola. Un approccio alternativo consiste nell'estendere alla logica del primo ordine la **fattorizzazione**, ovvero la rimozione di letterali ridondanti. Nella fattorizzazione proposizionale due letterali sono ridotti a uno se sono *identici*; in quella del primo ordine i letterali sono ridotti se sono *unificabili*. L'unificatore dev'essere applicato all'intera clausola. La combinazione di risoluzione binaria e fattorizzazione è completa.

risoluzione binaria

9.5.3 Alcuni esempi di dimostrazione

La risoluzione dimostra che $KB \models \alpha$ provando che $KB \wedge \neg\alpha$ non è soddisfacibile: per far questo deve derivare la clausola vuota. L'appuccio algoritmico è identico a quello del caso proposizionale, che abbiamo descritto nella Figura 7.13, ragion per cui non lo ripeteremo qui: al suo posto forniremo un paio di esempi. Il primo riguarda il colonnello criminale introdotto nel Paragrafo 9.3. Le formule in CNF sono:

$$\begin{aligned} & \neg Americano(x) \vee \neg Arma(y) \vee \neg Vende(x, y, z) \vee \neg Ostile(z) \vee Crimiale(x) \\ & \neg Missile(x) \vee \neg Possiede(Nono, x) \vee Vende(West, x, Nono) \\ & \neg Nemico(x, America) \vee Ostile(x) \\ & \neg Missile(x) \vee Arma(x) \\ & Possiede(Nono, M_1) \qquad \qquad \qquad Missile(M_1) \\ & Americano(West) \qquad \qquad \qquad Nemico(Nono, America). \end{aligned}$$

Dobbiamo anche includere l'obiettivo negato: $\neg Crimiale(West)$. La dimostrazione per risoluzione è mostrata nella Figura 9.10. Notate la sua struttura: una singola “spina dorsale”

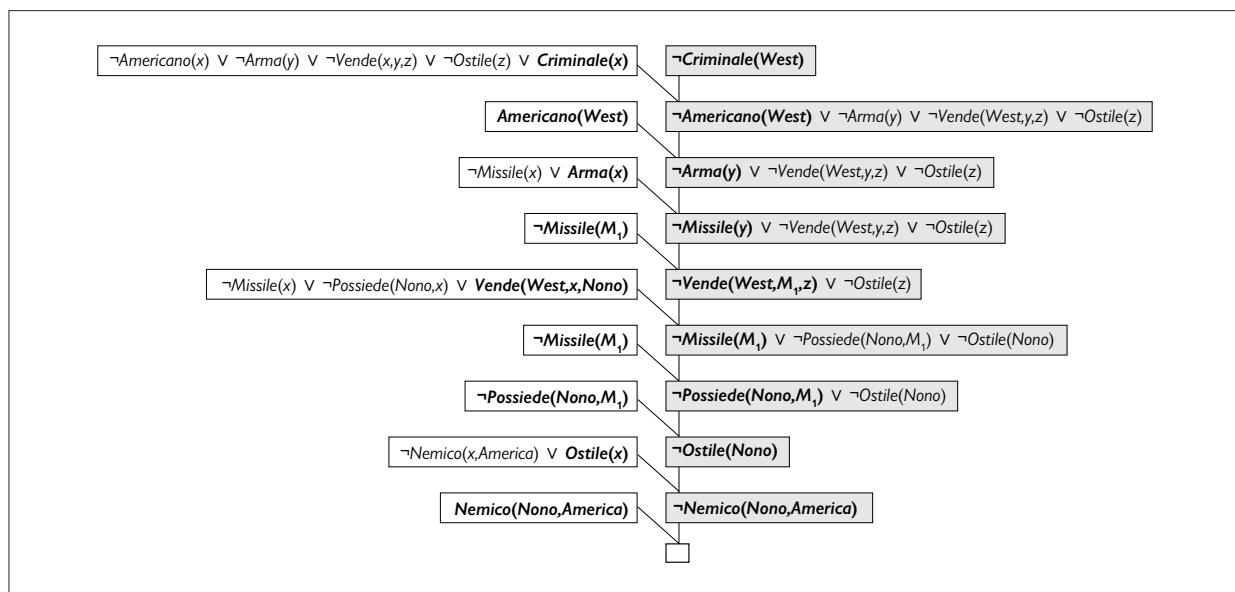


Figura 9.10 Una dimostrazione per risoluzione che il Colonnello West è un criminale. A ogni passaggio, i letterali che unificano sono evidenziati in grassetto e la clausola con il letterale positivo ha uno sfondo bianco.

di inferenze che comincia con la clausola obiettivo e continua a risolverla con clausole presenti nella base di conoscenza finché non viene generata la clausola vuota. Questa è la struttura caratteristica della risoluzione quando la base di conoscenza è composta da clausole di Horn. In effetti, le clausole lungo la linea principale corrispondono *esattamente* ai valori assunti consecutivamente dalla variabile *obiettivo* nell'algoritmo di concatenazione all'indietro dalla Figura 9.6. Questo è dovuto al fatto che abbiamo sempre scelto di risolvere con una clausola il cui letterale positivo si unifica con quello più a sinistra della clausola “corrente” sulla linea principale della dimostrazione; questo è esattamente ciò che accade nella concatenazione all'indietro. Possiamo quindi dire che la stessa concatenazione all'indietro è di fatto solo un caso speciale di risoluzione che adotta una particolare strategia di controllo per decidere l'ordine di esecuzione delle risoluzioni.

Il nostro secondo esempio utilizza la Skolemizzazione e coinvolge clausole che non sono definite. Questo fa sì che la struttura della dimostrazione sia un po’ più complessa. In linguaggio naturale il problema è il seguente:

Chiunque ami tutti gli animali è amato da qualcuno.
 Chiunque uccida un animale non è amato da nessuno.
 Jack ama tutti gli animali.
 O Jack o la Curiosità hanno ucciso il gatto, che si chiama Tonno.
 La Curiosità ha ucciso il gatto?

Prima di tutto esprimiamo le formule originali, un po’ di conoscenza iniziale e l’obiettivo G negato in logica del primo ordine:

- A. $\forall x [\forall y \text{Animale}(y) \Rightarrow \text{Ama}(x, y)] \Rightarrow [\exists y \text{ Ama}(y, x)]$
- B. $\forall x [\exists z \text{Animale}(z) \wedge \text{Uccide}(x, z)] \Rightarrow [\forall y \neg \text{Ama}(y, x)]$
- C. $\forall x \text{Animale}(x) \Rightarrow \text{Ama}(\text{Jack}, x)$
- D. $\text{Uccide}(\text{Jack}, \text{Tonno}) \vee \text{Uccide}(\text{Curiosità}, \text{Tonno})$
- E. $\text{Gatto}(\text{Tonno})$
- F. $\forall x \text{Gatto}(x) \Rightarrow \text{Animale}(x)$
- $\neg G. \neg \text{Uccide}(\text{Curiosità}, \text{Tonno})$

Ora applichiamo la procedura per convertire ogni formula in CNF:

- A1. $\text{Animale}(F(x)) \vee \text{Ama}(G(x), x)$
- A2. $\neg \text{Ama}(x, F(x)) \vee \text{Ama}(G(x), x)$
- B. $\neg \text{Ama}(y, x) \vee \neg \text{Animale}(z) \vee \neg \text{Uccide}(x, z)$
- C. $\neg \text{Animale}(x) \vee \text{Ama}(\text{Jack}, x)$
- D. $\text{Uccide}(\text{Jack}, \text{Tonno}) \vee \text{Uccide}(\text{Curiosità}, \text{Tonno})$
- E. $\text{Gatto}(\text{Tonno})$
- F. $\neg \text{Gatto}(x) \vee \text{Animale}(x)$
- $\neg G. \neg \text{Uccide}(\text{Curiosità}, \text{Tonno})$

La dimostrazione per risoluzione che la Curiosità ha ucciso il gatto è fornita nella Figura 9.11. In linguaggio naturale si potrebbe parafrasarla così:

Supponiamo che la Curiosità non abbia ucciso Tonno. Sappiamo che Jack o la Curiosità l'hanno fatto; quindi dev'essere stato Jack. Ora, Tonno è un gatto e i gatti sono animali, quindi anche Tonno è un animale. Dato che chiunque uccida un animale non è amato da nessuno, sappiamo che nessuno ama Jack. D'altra parte, Jack ama tutti gli animali, per cui qualcuno lo ama; quindi abbiamo una contraddizione. Ne conseguue che è stata la Curiosità a uccidere il gatto.

Questa dimostrazione risponde alla domanda “La Curiosità ha ucciso il gatto?”, ma spesso vorremmo porre domande più generali, come “Chi ha ucciso il gatto?”. La risoluzione può rispondere anche a queste interrogazioni, ma il compito è più faticoso. L’obiettivo è $\exists w$

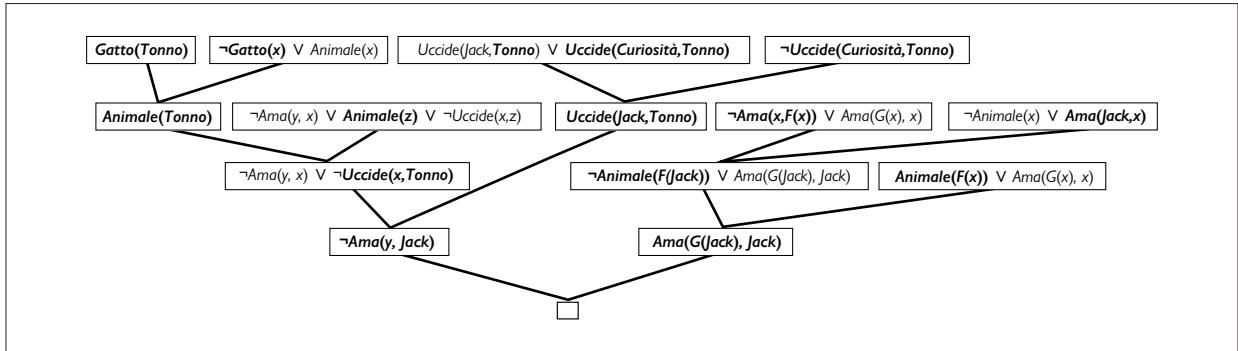


Figura 9.11 Una dimostrazione per risoluzione che la Curiosità ha ucciso il gatto. Notate l’uso della fattorizzazione nella derivazione della clausola $Ama(G(jack), Jack)$. Notate anche, in alto a destra, che l’unificazione di $Ama(x, F(x))$ e $Ama(Jack, x)$ può avere successo soltanto dopo che le variabili sono state standardizzate separatamente.

$Uccide(w, Tonno)$ che, una volta negato, diventa in CNF $\neg Uccide(w, Tonno)$. Ripetendo la dimostrazione della Figura 9.11 con il nuovo obiettivo negato otteniamo un albero simile, che comprende in uno dei passi la sostituzione $\{w/Curiosità\}$. In questo caso, quindi, appurare chi ha ucciso il gatto è solo questione di tener traccia dei legami delle variabili comprese nell’interrogazione. Sfortunatamente, per obiettivi esistenziali la risoluzione può talvolta produrre **dimostrazioni non costruttive**, in cui sappiamo che un’interrogazione è vera, ma non c’è un legame unico per la variabile.

dimostrazione non costruttiva

9.5.4 Completezza della risoluzione

In questo paragrafo dimostriamo la completezza della risoluzione. Se siete disposti a crederci sulla parola, potete saltarlo senza problemi.

Mostreremo che la procedura è **completa per refutazione**, il che significa che se un insieme di formule è insoddisfacibile la risoluzione sarà sempre in grado di derivare una contraddizione. L’algoritmo non può generare tutte le conseguenze logiche di un insieme di formule, ma può sempre stabilire che una data formula è conseguenza logica di un insieme di formule. Di conseguenza può essere usato per trovare tutte le risposte a una specifica domanda, $Q(x)$, dimostrando che $KB \wedge \neg Q(x)$ è insoddisfacibile.

completezza per refutazione

Considereremo assodato che ogni formula in logica del primo ordine (che non includa l’uguaglianza) può essere riscritta come un insieme di clausole in CNF. Questo può essere dimostrato per induzione usando le formule atomiche come caso base (Davis e Putnam, 1960). Il nostro obiettivo quindi è dimostrare quanto segue: *se S è un insieme di clausole non soddisfacibile, l’applicazione su S di un numero finito di passi di risoluzione porterà a una contraddizione*.

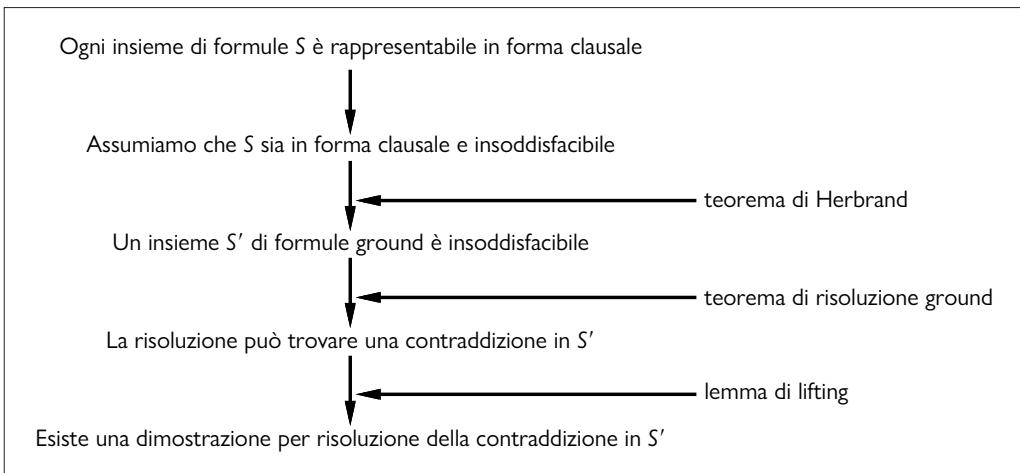


Il nostro schema di dimostrazione segue quello originale di Robinson, con qualche semplificazione introdotta da Genesereth e Nilsson (1987). La struttura base della dimostrazione è mostrata nella Figura 9.12 e procede come segue.

1. Per prima cosa osserviamo che, se S non è soddisfacibile, esiste un particolare insieme di *istanze ground* delle clausole di S anch’esso insoddisfacibile (teorema di Herbrand).
2. Facciamo poi ricorso al **teorema di risoluzione ground** già presentato nel Capitolo 7, che afferma che la risoluzione proposizionale è completa per le formule ground.
3. Applichiamo poi il **lemma di lifting** per mostrare che, per ogni dimostrazione con risoluzione proposizionale che usa un insieme di formule ground, esiste una corrispondente dimostrazione che usa le formule del primo ordine da cui sono state ricavate le formule ground originarie.

Figura 9.12

Struttura della dimostrazione di completezza della risoluzione.



Per compiere il primo passo dobbiamo introdurre tre concetti nuovi.

universo di Herbrand

- **Universo di Herbrand:** se S è un insieme di clausole allora H_S , l'universo di Herbrand di S , è costituito dall'insieme di tutti i termini ground che si possono costruire a partire da:
 - I simboli di funzione in S , se ve ne sono.
 - I simboli di costante in S , se ve ne sono; in caso contrario, un simbolo di costante di default, S .

Per esempio, se S contiene solo la clausola $\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$, H_S è costituito dal seguente insieme infinito di termini ground:

$$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \dots\}.$$

saturazione

- **Saturazione:** se S è un insieme di clausole e P è un insieme di termini ground allora $P(S)$, la saturazione di S rispetto a P , è l'insieme delle clausole ground ottenute applicando tutte le possibili sostituzioni consistenti di termini ground in P alle variabili in S .

base di Herbrand

- **Base di Herbrand:** la saturazione di un insieme S di clausole rispetto al suo universo di Herbrand prende il nome di base di Herbrand di S e si scrive $H_S(S)$. Per esempio, se S contiene solo la clausola che abbiamo scritto qui sopra, $H_S(S)$ sarà costituito dall'insieme infinito di clausole:

$$\begin{aligned} &\{\neg P(A, F(A, A)) \vee \neg Q(A, A) \vee R(A, B), \\ &\quad \neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B), \\ &\quad \neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B), \\ &\quad \neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \dots\}. \end{aligned}$$

teorema di Herbrand

Queste definizioni ci permettono di formulare una versione del **teorema di Herbrand** (Herbrand, 1930):

Se un insieme S di clausole è insoddisfacibile, allora esiste un sottoinsieme finito di $H_S(S)$ anch'esso insoddisfacibile.

Chiamiamo S' questo sottoinsieme finito di formule ground. Ora possiamo sfruttare il teorema di risoluzione ground (cfr. paragrafo 7.5.2) per dimostrare che la **chiusura della risoluzione** $RC(S')$ contiene la clausola vuota: questo significa che eseguire la risoluzione proposizionale su S' fino al completamento deriverà una contraddizione.

Ora che abbiamo stabilito che esiste sempre una dimostrazione per risoluzione che richiede un sottoinsieme finito della base di Herbrand di S , il passo successivo è mostrare che esiste una dimostrazione per risoluzione che usa le clausole di S stesso, che non sono neces-



IL TEOREMA DI INCOMPLETEZZA DI GÖDEL

Estendendo leggermente il linguaggio della logica del primo ordine in modo da consentire l'uso dello **schema di induzione matematica** nell'aritmetica, Kurt Gödel è stato in grado di mostrare, nel suo **teorema di incompletezza**, che esistono formule aritmetiche vere che non possono essere dimostrate.

La dimostrazione del teorema di incompletezza va decisamente oltre lo scopo di questo libro, dato che è lunga almeno 30 pagine, ma possiamo farvi un accenno. Cominciamo dalla formulazione, in logica, della teoria dei numeri: in questa teoria c'è una sola costante, 0, e una sola funzione, S (la funzione successore). Nel modello inteso, $S(0)$ indica 1, $S(S(0))$ 2, e così via; il linguaggio quindi ha un nome per ogni numero naturale. Il vocabolario include anche i simboli di funzione $+$, \times e Expt (elevamento a potenza) nonché il consueto insieme di connettivi logici e quantificatori. Il primo passo è notare che l'insieme di formule che possiamo scrivere può essere enumerato: immaginate infatti di definire un ordine alfabetico per i simboli e poi scrivere in ordine alfabetico le formule di lunghezza 1, poi quelle di lunghezza 2 e così via. A questo punto sarà possibile numerare ogni formula α con un numero naturale distinto $\# \alpha$ (il **numero di Gödel**). Questo punto è cruciale: la teoria dei numeri contiene un nome per ognuna delle sue formule. In modo analogo possiamo numerare ogni possibile dimostrazione P con un numero di Gödel $G(P)$, perché una dimostrazione non è altro che una sequenza finita di formule.

Ora supponiamo di avere un insieme ricorsivamente enumerabile A di formule che rappresentano affermazioni vere riguardanti i numeri naturali. Ricordando che alle formule di A si può fare riferimento per mezzo di un dato insieme di interi, possiamo immaginare di scrivere in linguaggio naturale una formula $\alpha(j, A)$ come la seguente:

$\forall i \ i \text{ non è il numero di Gödel di una dimostrazione della formula il cui numero di Gödel è } j, \text{ ove la dimostrazione sfrutta solo premesse contenute in } A.$

Sia poi σ la formula $\alpha(\# \sigma, A)$, ovvero la formula che afferma la propria stessa indimostrabilità da A (questa formula esiste sempre, anche se la cosa può non apparire evidente).

Ora possiamo formulare la seguente, ingegnosa argomentazione: supponiamo che σ sia effettivamente dimostrabile partendo da A ; allora σ è falsa (dato che σ stessa afferma che non può essere dimostrata). Ma allora abbiamo una formula falsa dimostrabile da A , il che significa che A non può consistere solo di formule vere, cosa che costituisce una violazione della nostra premessa. Ne consegue che σ non è dimostrabile partendo da A . Ma questo è esattamente ciò che sostiene σ stessa; quindi σ è una formula vera.

In questo modo abbiamo dimostrato (a meno di altre 29 pagine e mezza) che per ogni insieme di formule vere della teoria dei numeri, e in particolare per ogni insieme di assiomi base, ci sono altre formule vere che non possono essere dimostrate partendo da tali assiomi. Questo significa, tra l'altro, che non potremo mai dimostrare tutti i teoremi della matematica *dato un qualsiasi sistema di assiomi*. Chiaramente questa scoperta è stata molto importante per la matematica: il suo significato per quanto riguarda l'IA è stato oggetto di un acceso dibattito, a partire da alcune riflessioni di Gödel stesso. Ci occuperemo dell'argomento nel Capitolo 27.

sariamente ground. Cominceremo con il considerare una singola applicazione della regola di risoluzione. Robinson ha stabilito il seguente lemma:

Siano C_1 e C_2 due clausole senza variabili in comune, e siano C'_1 e C'_2 istanze ground di C_1 e C_2 . Se C' è risolvente di C'_1 e C'_2 , allora esiste una clausola C tale che (1) C è risolvente di C_1 e C_2 e (2) C' è un'istanza ground di C .

Questo è chiamato **lemma di lifting**, perché “solleva” un passo di dimostrazione da clausole ground a clausole generali del primo ordine. Per dimostrare il lemma di lifting, Robinson

lemma di lifting

dovette inventare l'unificazione e derivare tutte le proprietà degli unificatori più generali. Invece di ripetere qui la dimostrazione, ci limiteremo a illustrare il lemma:

$$\begin{aligned} C_1 &= \neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B) \\ C_2 &= \neg N(G(y), z) \vee P(H(y), z) \\ C'_1 &= \neg P(H(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C'_2 &= \neg N(G(B), F(H(B), A)) \vee P(H(B), F(H(B), A)) \\ C' &= \neg N(G(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C &= \neg N(G(y), F(H(y), A)) \vee \neg Q(H(y), A) \vee R(H(y), B). \end{aligned}$$

Vediamo che C' è effettivamente un'istanza ground di C . In generale, affinché C'_1 e C'_2 abbiano dei risolventi devono essere costruite applicando per prima cosa a C_1 e C_2 l'unificatore più generale di una coppia di letterali complementari in C_1 e C_2 . Dal lemma di lifting è facile derivare un'asserzione simile riguardo a una sequenza qualsiasi di applicazioni della regola di risoluzione:

Per ogni clausola C' nella chiusura della risoluzione di S' c'è una clausola C nella chiusura della risoluzione di S tale che C' sia un'istanza ground di C e che la derivazione di C abbia la stessa lunghezza di quella di C'

Da questo fatto segue che se la clausola vuota compare nella chiusura della risoluzione di S' deve comparire anche nella chiusura della risoluzione di S : infatti la clausola vuota non può essere un'istanza ground di alcuna altra clausola. Ricapitolando, abbiamo dimostrato che se S non è soddisfacibile, allora esiste una derivazione finita della clausola vuota mediante risoluzione.

Il lifting da clausole ground a clausole del primo ordine rappresenta un grande incremento di potenza nella dimostrazione dei teoremi. Questo deriva dal fatto che nella logica del primo ordine si devono istanziare variabili solo quando la dimostrazione lo richiede, mentre i metodi basati su clausole ground devono esaminare un numero enorme di istanziazioni arbitrarie.

9.5.5 L'uguaglianza

Nessuno dei metodi di inferenza descritti fin qui è in grado di gestire un'asserzione della forma $x = y$ senza lavoro aggiuntivo. Si possono adottare tre approcci distinti. Il primo consiste nell'assiomatizzare l'uguaglianza, aggiungendo alla base di conoscenza formule che la riguardano. Si deve specificare che è riflessiva, simmetrica e transitiva, e dobbiamo anche dire che in ogni predicato o funzione possiamo sostituire elementi uguali. Sono necessari quindi tre assiomi base, più uno per ogni predicato e funzione:

$$\begin{aligned} \forall x \ x &= x \\ \forall x, y \ x &= y \Rightarrow y = x \\ \forall x, y, z \ x &= y \wedge y = z \Rightarrow x = z \\ \forall x, y \ x &= y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\ \forall x, y \ x &= y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\ &\vdots \\ &\vdots \\ \forall w, x, y, z \ w &= y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z)) \\ \forall w, x, y, z \ w &= y \wedge x = z \Rightarrow (F_2(w, x) = F_2(y, z)) \\ &\vdots \end{aligned}$$

Date queste formule, una procedura di inferenza standard come la risoluzione può eseguire compiti che richiedono di ragionare sull'uguaglianza, come il calcolo di equazioni matematiche.

che. Tuttavia, questi assiomi genereranno una grande quantità di conclusioni, molte delle quali inutili per una dimostrazione. Il secondo approccio consiste nell'aggiungere regole di inferenza anziché assiomi. La regola più semplice è la **demodulazione**, che prende una clausola unitaria $x = y$ e una clausola α che contiene il termine x , e restituisce una nuova clausola ottenuta sostituendo y al posto di x in α . Funziona se il termine in α si unifica con x ; non deve essere esattamente uguale a x . Ciò significa che la demodulazione può essere utilizzata per semplificare espressioni mediante demodulatori quali $z + 0 = z$ o $z^1 = z$. Come altro esempio, dati:

$$\begin{aligned} \text{Padre}(\text{Padre}(x)) &= \text{NonnoPaterno}(x) \\ \text{Annonascita}(\text{Padre}(\text{Padre}(\text{Bella})), 1926) \end{aligned}$$

possiamo concludere, per demodulazione:

$$\text{Annonascita}(\text{NonnoPaterno}(\text{Bella}), 1926).$$

Più formalmente,abbiamo:

- **demodulazione:** dati tre termini x, y e z , dove z appare nel letterale m_i e $\text{UNIFY}(x, z) = \theta$: **demodulazione**

$$\frac{x = y, \quad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), m_1 \vee \cdots \vee m_n)}.$$

dove SUBST è la consueta sostituzione di una lista di legami e $\text{SUB}(x, y, m)$ significa sostituire x con y in qualche punto di m .

La regola può essere estesa per gestire anche clausole non unitarie in cui appare un segno di uguaglianza:

- **paramodulazione:** dati tre termini x, y e z , dove z appare nel letterale m_i e $\text{UNIFY}(x, z) = \theta$, **paramodulazione**

$$\frac{\ell_1 \vee \cdots \vee \ell_k \vee x = y, \quad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), \text{SUBST}(\theta, \ell_1 \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_n))}.$$

Per esempio, da:

$$P(F(x, B), x) \vee Q(x) \quad \text{e} \quad F(A, y) = y \vee R(y)$$

abbiamo $\theta = \text{UNIFY}(F(A, y), (F(x, B))) = \{x/A, y/B\}$, e possiamo ottenere per paramodulazione la formula:

$$P(B, A) \vee Q(A) \vee R(B).$$

La paramodulazione produce una procedura di inferenza completa per la logica del primo ordine comprensiva di uguaglianza.

Un terzo approccio delega completamente il ragionamento sull'uguaglianza a un algoritmo esteso di unificazione. Questo significa che i termini sono unificabili se è *dimostrabile* che sono uguali sotto qualche sostituzione, dove “dimostrabile” permette un certo grado di ragionamento sull'uguaglianza. Per esempio, i termini $1 + 2$ e $2 + 1$ normalmente non sono unificabili, ma un algoritmo di unificazione che sa che $x + y = y + x$ li può unificare con una sostituzione vuota. Una **unificazione equazionale** di questo tipo può essere svolta mediante algoritmi efficienti progettati ad hoc per i particolari assiomi utilizzati (commutatività, associatività e così via) anziché ricorrere all'inferenza esplicita tramite gli assiomi stessi. I dimostratori di teoremi che sfruttano questa tecnica sono strettamente imparentati ai sistemi di programmazione logica che abbiamo descritto nel Paragrafo 9.4.

**unificazione
equazionale**

9.5.6 Strategie di risoluzione

Abbiamo assodato che una serie di applicazioni della regola di risoluzione porterà a scoprire una dimostrazione, se questa esiste. Ora esamineremo alcune strategie che aiutano a trovare una dimostrazione *in modo efficiente*.

preferenza per le clausole unitarie**Preferenze per le clausole unitarie**

Questa strategia preferisce applicare la risoluzione quando una delle due formule è un letterale singolo (noto anche come **clausola unitaria**). Il concetto si basa sul fatto che stiamo comunque cercando di produrre una clausola vuota, ragion per cui è una buona idea preferire le inferenze che producono clausole più corte. Risolvere una formula unitaria (come P) con qualsiasi altra formula (come $\neg P \vee \neg Q \vee R$) produce sempre una clausola (in questo caso $\neg Q \vee R$) più corta di quella originale. La prima volta che fu applicata al caso proposizionale, nel 1964, la strategia di preferenza per le clausole unitarie portò a un incremento delle prestazioni davvero notevole, rendendo possibile dimostrare teoremi che in caso contrario non avrebbero potuto essere trattati. La **risoluzione unitaria** è una forma ristretta di risoluzione in cui ogni passo deve obbligatoriamente coinvolgere una clausola unitaria. In generale non è completa, ma lo diventa per le clausole di Horn: in quest'ultimo caso assomiglia a una concatenazione in avanti.

Il dimostratore di teoremi OTTER (McCune, 1992) utilizza una forma di ricerca best-first. La sua funzione euristica misura il “peso” di ogni clausola, preferendo le clausole più leggere. La scelta esatta dell’euristica spetta all’utente, ma in generale il peso di una clausola dovrebbe essere correlato alla sua dimensione o difficoltà. Le clausole unitarie sono considerate leggere; la ricerca può dunque essere considerata come una generalizzazione della strategia di preferenza per le clausole unitarie.

insieme di supporto**Insieme di supporto**

Le euristiche che consentono di provare prima certe tipologie di risoluzione sono utili, ma in generale è più efficace cercare di eliminarne alcune del tutto. Per esempio, possiamo imporre che ogni passo di risoluzione coinvolga almeno un elemento di uno speciale insieme di clausole, l'*insieme di supporto*. La clausola risolvente verrà quindi aggiunta all’insieme di supporto. Se l’insieme è piccolo rispetto all’intera base di conoscenza, lo spazio di ricerca risulterà notevolmente ridotto.

Per garantire la completezza di questa strategia, possiamo scegliere l’insieme di supporto S in modo tale che le formule restanti siano collettivamente soddisfacibili. Per esempio, si può scegliere come insieme di supporto la query negata, partendo dall’assunto che la base di conoscenza originale sia consistente (del resto, se così non fosse avrebbe poco senso dimostrare che la query è conseguenza logica). Questa strategia ha il vantaggio aggiuntivo di generare alberi di dimostrazione guidati dall’obiettivo che spesso sono particolarmente leggibili per gli esseri umani.

risoluzione di input**Risoluzione di input**

Nella strategia di risoluzione di input, ogni risoluzione combina una formula di input (dalla KB o dalla query) con un’altra. La dimostrazione riportata nella Figura 9.10 usa solo risoluzioni di input e ha la caratteristica struttura formata da una “spina dorsale” a cui si attaccano singole formule. È chiaro che lo spazio degli alberi di dimostrazione che hanno questa struttura è più piccolo dello spazio di tutti i grafi di dimostrazione. Nelle basi di conoscenza di Horn, il Modus Ponens è un tipo di strategia di risoluzione di input, perché combina un’implicazione dalla KB originale con qualche altra formula. Non sorprende quindi che la risoluzione di input sia completa per le basi di conoscenza in forma di Horn, ma incompleta nel caso generale. La strategia di **risoluzione lineare** è una lieve generalizzazione che permette di risolvere insieme P e Q se P appartiene alla KB originale oppure se P è un antenato di Q nell’albero di dimostrazione. La risoluzione lineare è completa.

risoluzione lineare**sussunzione****Sussunzione**

Il metodo della sussunzione elimina tutte le formule che sono sussunte da (ovvero più specifiche di) una formula esistente nella KB. Per esempio, se $P(x)$ è nella KB, non c’è alcuna

ragione di aggiungere $P(A)$ e ha ancor meno senso aggiungere $P(A) \vee Q(B)$. La sussunzione aiuta a mantenere piccola la KB e lo spazio di ricerca.

Apprendimento

apprendimento

È possibile migliorare un dimostratore di teoremi mediante l'apprendimento dall'esperienza. Dato un insieme di teoremi già dimostrati, si può addestrare un sistema di apprendimento automatico a rispondere alla domanda: dati un insieme di premesse e un obiettivo da dimostrare, quali passaggi di dimostrazione sono simili a passaggi che hanno già avuto successo in passato? È proprio quello che fa il sistema DEEPOL (Bansal et. al., 2019), usando reti neurali deep (cfr. Capitolo 21 del Volume 2) per costruire modelli (denominati *embedding*) di obiettivi e premesse e servendosi di questi per effettuare selezioni. Per l'addestramento si possono usare come esempi dimostrazioni sia umane sia generate dal computer, partendo da una raccolta di 10.000 dimostrazioni.

Utilizzi pratici dei dimostratori di teoremi

Abbiamo mostrato che la logica del primo ordine può rappresentare un semplice scenario del mondo reale con concetti quali vendita, armi e cittadinanza. In scenari reali più complessi, tuttavia, ci sono troppa incertezza e troppe incognite. L'utilizzo della logica si è dimostrato più efficace per scenari con concetti formali e ben definiti, come la **sintesi** e la **verifica** di hardware e software. La ricerca sulla dimostrazione automatica di teoremi, quindi, oltre che all'IA si estende anche al campo della progettazione hardware, dei linguaggi di programmazione e dell'ingegneria del software.

sintesi
verifica

Nel caso dell'hardware, gli assiomi descrivono le interazioni tra segnali ed elementi di un circuito (cfr. Paragrafo 8.4.2 per un esempio). Sistemi di ragionamento logico progettati appositamente sono stati in grado di verificare la correttezza di intere CPU, incluse le loro proprietà temporali (Sivas e Bickford, 1990). Il dimostratore di teoremi AURA è stato applicato nella progettazione di circuiti e ha prodotto circuiti più compatti rispetto a qualsiasi progetto precedente (Wojciechowski e Wojcik, 1983).

Nel caso del software, ragionare sui programmi è abbastanza simile a ragionare su azioni, come nel Capitolo 7: gli assiomi descrivono le precondizioni e gli effetti di ogni istruzione. La sintesi formale di algoritmi fu una delle prime applicazioni dei dimostratori di teoremi, come delineato da Cordell Green (1969a), che ha esteso idee precedenti di Simon (1963). L'idea è dimostrare costruttivamente un teorema della forma “esiste un programma p che soddisfa una determinata specifica”. Benché una versione completamente automatizzata di quella che è stata chiamata **sintesi deduttiva** non sia ancora disponibile per la programmazione in generale, con l'aiuto di utenti umani la sintesi deduttiva è stata in grado di progettare con successo diversi algoritmi sofisticati e innovativi. La sintesi di programmi di uso speciale, come il codice per il calcolo scientifico, è un'altra area in cui la ricerca è molto attiva.

Tecniche simili cominciano ora a essere applicate alla verifica software, grazie a sistemi come il model checker SPIN (Holzmann, 1997). Il programma di controllo Remote Agent per veicoli spaziali, per esempio, è stato sottoposto a verifica sia prima che dopo il volo (Havelund et al., 2000). L'algoritmo di cifratura a chiave pubblica RSA e l'algoritmo per il matching fra stringhe di Boyer-Moore sono stati verificati in questo modo (Boyer e Moore, 1984).

9.6 Riepilogo

Abbiamo presentato un’analisi dell’inferenza nella logica del primo ordine e diversi algoritmi per eseguirla.

- Un primo approccio consiste nell’usare regole di inferenza (**istanziazione universale** e **istanziazione esistenziale**) per **proposizionalizzare** il problema. Tipicamente questa tecnica risulta lenta, a meno che il dominio sia piccolo.
- L’uso dell’**unificazione** per identificare le sostituzioni appropriate delle variabili elimina il passo di istanziazione nelle dimostrazioni del primo ordine, rendendo il processo più efficiente in molti casi.
- Una versione del Modus Ponens “sollevata” attraverso il lifting usa l’unificazione per dare origine a una regola di inferenza intuitiva e potente, il **Modus Ponens generalizzato**. Gli algoritmi di **concatenazione in avanti** e **concatenazione all’indietro** applicano questa regola a insiemi di clausole definite.
- Il Modus Ponens generalizzato è completo per le clausole definite, benché il problema della conseguenza logica rimanga **semidecidibile**. La conseguenza logica è comunque decidibile nel caso di basi di conoscenza **Datalog**, che consistono di clausole definite prive di simboli di funzione.
- La concatenazione in avanti è usata nei **databases deduttivi**, in cui può essere combinata con le operazioni classiche dei database relazionali. È utilizzata anche nei **sistemi di produzioni**, che eseguono aggiornamenti efficienti anche in presenza di insiemi di regole molto grandi. La concatenazione in avanti è completa per Datalog e richiede un tempo di esecuzione polinomiale.
- La concatenazione all’indietro è usata nei **sistemi di programmazione logica**, che sfruttano sofisticati compilatori per ottenere un’inferenza molto veloce. Soffre dei problemi di inferenze ridondanti e cicli infiniti.
- Il **Prolog**, a differenza della logica del primo ordine, utilizza un mondo chiuso con l’ipotesi dei nomi unici e la negazione come fallimento. Questo ne fa un linguaggio di programmazione più pratico, ma più lontano dalla logica pura.
- La regola di inferenza della **risoluzione** rappresenta un sistema completo di dimostrazione per la logica del primo ordine, applicabile a basi di conoscenza in forma normale congiuntiva.
- Esistono molte strategie per ridurre lo spazio di ricerca di un sistema di risoluzione senza comprometterne la completezza. Uno dei problemi più importanti è la gestione dell’ugualianza; abbiamo mostrato come utilizzare la **demodulazione** e la **paramodulazione**.
- Dimostratori di teoremi efficienti sono stati utilizzati per dimostrare interessanti teoremi matematici e per verificare e sintetizzare software e hardware.

Note storiche e bibliografiche

Gottlob Frege, che sviluppò la logica del primo ordine nella sua prima versione completa nel 1879, basò il suo sistema di inferenza su una collezione di schemi logicamente validi più una singola regola di inferenza, il Modus Ponens. Whitehead e Russell (1910) formularono le cosiddette *regole di passaggio* (il termine è preso da Herbrand (1930)) per spostare i quantifica-

tori nella parte iniziale delle formule. Le costanti e le funzioni di Skolem furono introdotte, come prevedibile, da Thoralf Skolem (1920). La procedura generale di skolemizzazione è ancora dovuta a Skolem (1928), così come (stranamente) l’importante nozione di universo di Herbrand.

Il teorema di Herbrand (1930) ha giocato un ruolo fondamentale nello sviluppo dei metodi di ragionamento automatico. Herbrand può anche essere considerato l'inventore dell'**unificazione**. Gödel (1930) partì dalle idee di Skolem e Herbrand per dimostrare che la logica del primo ordine ha una procedura di dimostrazione completa. Alan Turing (1936) e Alonzo Church (1936) hanno dimostrato simultaneamente, usando metodi molto diversi, che nella logica del primo ordine la validità non è decidibile. L'eccellente volume di Enderton (1972) espone tutti questi risultati in modo rigoroso ma abbastanza comprensibile.

Abraham Robinson sostenne che un sistema di ragionamento automatico potesse essere realizzato utilizzando la proposizionalizzazione e il teorema di Herbrand, mentre Paul Gilmore (1960) introdusse il metodo di proposizionalizzazione del Paragrafo 9.1. Prawitz (1960) sviluppò l'idea chiave di generare termini dall'universo di Herbrand solo quando era necessario per stabilire un'inconsistenza proposizionale. Dopo ulteriori sviluppi da parte di altri ricercatori, quest'idea portò John Alan Robinson (nessuna parentela) a sviluppare il metodo di risoluzione (Robinson, 1965).

La risoluzione fu adottata per sistemi di domanda-risposta da Cordell Green e Bertram Raphael (1968). Le prime implementazioni nel campo dell'IA dedicarono molti sforzi alla creazione di strutture dati che permettessero un recupero efficiente dei fatti; i frutti di questo lavoro sono presentati nei testi di programmazione per l'IA (Charniak *et al.*, 1987; Norvig, 1992; Forbus e de Kleer, 1993). All'inizio degli anni 1970 la **concatenazione in avanti** si era affermata come un'alternativa alla risoluzione di più facile comprensione. Solitamente le applicazioni di IA prevedevano un grande numero di regole, cosa che rendeva importante disporre di una tecnologia di matching delle regole efficiente, in particolar modo per gli aggiornamenti incrementali. Per supportare applicazioni simili furono sviluppati i **sistemi di produzioni**. Il linguaggio per sistemi di produzioni OPS-5 (Forgy, 1981; Brownston *et al.*, 1985), che incorporava l'efficiente processo di matching **rete** (Forgy, 1982), fu utilizzato applicazioni quali il sistema esperto R1 per la configurazione di minicomputer (McDermott, 1982). Kraska *et al.* (2017) spiegano come le reti neurali possano apprendere uno schema di indicizzazione efficiente per specifici insiemi di dati.

L'architettura cognitiva SOAR (Laird *et al.*, 1987; Laird, 2008) fu progettata per gestire insiemi di numerosissime regole, fino a un milione (Doorenbos, 1984).

Tra le sue applicazioni vi sono il controllo di velivoli da combattimento simulati (Jones *et al.*, 1998), la gestione di veicoli spaziali (Taylor *et al.*, 2007), personaggi IA per giochi al computer (Wintermute *et al.*, 2007), strumenti per l'addestramento dei soldati (Wray e Jones, 2005).

Il campo dei **database deduttivi** ha avuto inizio con un workshop a Tolosa nel 1977, che vide la partecipazione di diversi esperti di inferenza logica e database (Gallaire e Minker, 1978). Importanti lavori di Chandra e Harel (1980) e di Ullman (1985) portarono all'adozione di **Datalog** come linguaggio standard per i database deduttivi. Lo sviluppo, da parte di Bancilhon *et al.* (1986), della tecnica dei *magic set* per la riscrittura di regole permise poi alla concatenazione in avanti di "prendere in prestito" da quella all'indietro il vantaggio di essere guidata dall'obiettivo.

La crescita di Internet ha portato a una maggiore disponibilità di grandi database online, e questo ha acceso l'interesse nell'integrazione di più database in uno spazio dati consistente (Halevy, 2007). Kraska *et al.* (2017) hanno mostrato incrementi di velocità fino al 70% usando tecniche di apprendimento automatico per creare **strutture di indicizzazione apprese** che favoriscono una ricerca efficiente dei dati.

La **concatenazione all'indietro** per l'inferenza logica ha avuto origine nel linguaggio PLANNER (Hewitt, 1969). Intanto, nel 1972, Alain Colmerauer aveva sviluppato e implementato **Prolog** con lo scopo di analizzare la sintassi del linguaggio naturale: in effetti le clausole Prolog inizialmente rappresentavano regole di una grammatica non contestuale (Roussel, 1975; Colmerauer *et al.*, 1973).

Gran parte dei fondamenti teorici della programmazione logica furono sviluppati da Robert Kowalski all'Imperial College di Londra, insieme a Colmerauer; cfr. Kowalski (1988) e Colmerauer e Roussel (1993) per una panoramica storica. I compilatori Prolog più efficienti sono generalmente basati sul modello di computazione della Warren Abstract Machine (WAM), sviluppato da David H. D. Warren (1983). Van Roy (1990) ha mostrato che i programmi Prolog possono essere competitivi con quelli scritti in C in termini di velocità.

Metodi per evitare cicli superflui nei programmi logici ricorsivi furono sviluppati indipendentemente da Smith *et al.* (1986) e Tamaki e Sato (1986). Quest'ultimo lavoro includeva anche il concetto di memoizzazione per i programmi logici, in seguito sviluppato estensivamente da David S. Warren sotto il nome di **programmazione logica con tabelle**. Swift e Warren

(1994) mostraron come estendere la WAM per gestire tabelle, permettendo ai programmi Datalog di girare un ordine di grandezza più velocemente dei database deduttivi basati sulla concatenazione in avanti.

Le prime ricerche sulla programmazione logica a vincoli furono svolte da Jaffar e Lassez (1987). Jaffar *et al.* svilupparono il sistema CLP(R) per gestire vincoli a valori reali. Oggi esistono prodotti commerciali per risolvere problemi di configurazione e ottimizzazione su larga scala mediante la programmazione a vincoli; uno dei più noti è ILOG (Junker, 2003). L'*answer set programming* (Gelfond, 2008) estende il Prolog consentendo disgiunzione e negazione.

Tra i testi che trattano programmazione logica e Prolog citiamo Shoham (1994), Bratko (2009), Clocksin (2003), Clocksin e Mellish (2003). Prima del 2000, la rivista più importante era il *Journal of Logic Programming*; è stata rimpiazzata da *Theory and Practice of Logic Programming*. I congressi sulla programmazione logica includono l'International Conference on Logic Programming (ICLP) e l'International Logic Programming Symposium (ILPS).

La ricerca nel campo della **dimostrazione di teoremi matematici** era cominciata ancor prima che fossero sviluppati i primi sistemi completi del primo ordine. Il Geometry Theorem Prover di Herbert Gelernter (Gelernter, 1959) usava metodi di ricerca euristica uniti a diagrammi per la potatura di sotto-oggettivi falsi e fu in grado di dimostrare risultati alquanto complessi di geometria euclidea. Le regole di **demodulazione e paramodulazione** per il ragionamento sull'uguaglianza furono introdotte rispettivamente da Wos *et al.* (1967) e Wos e Robinson (1968). Queste regole furono anche sviluppate indipendentemente nel contesto dei sistemi di riscrittura di termini (Knuth e Bendix, 1970). L'incorporazione del ragionamento sull'uguaglianza nell'algoritmo di unificazione è dovuta a Gordon Plotkin (1972). Jouannaud e Kirchner (1991) forniscono una panoramica sull'unificazione equazionale dal punto di vista della riscrittura di termini. Una panoramica sull'unificazione è offerta da Baader e Snyder (2001).

Per la risoluzione sono state proposte molte strategie di controllo, a partire da quella che preferisce le clausole unitarie (Wos *et al.*, 1964). La strategia basata su un insieme di supporto è stata proposta da Wos *et al.* (1965) in modo da permettere alla risoluzione di essere guidata in una certa misura dall'obiettivo. La risoluzione lineare è apparsa per la prima volta in Loveland (1970). Genesereth e Nilsson (1987, Capitolo 5) offrono un'analisi di una grande varietà di strategie

di controllo. Alemi *et al.* (2017) mostrano come il sistema DEEPMATH utilizzi reti neurali deep per selezionare gli assiomi con maggiore probabilità di condurre a una dimostrazione una volta immessi in un dimostratore di teoremi tradizionale. In un certo senso, la rete neurale assume il ruolo dell'intuito matematico, mentre il dimostratore di teoremi assume il ruolo della competenza tecnica matematica. Loos *et al.* (2017) mostrano che questo approccio può essere esteso per guidare la ricerca e consentire così di dimostrare un maggior numero di teoremi.

A *Computational Logic* (Boyer e Moore, 1979) è il riferimento principale sul dimostratore di teoremi di Boyer-Moore. Stickel (1988) descrive il Prolog Technology Theorem Prover (PTTP), che unisce la compilazione Prolog con l'eliminazione dei modelli. SETHEO (Letz *et al.*, 1992) è un altro dimostratore di teoremi ampiamente utilizzato che si basa sullo stesso approccio. LEANTAP (Beckert e Posegga, 1995) è un efficiente dimostratore di teoremi implementato in sole 25 righe di Prolog. Weidenbach (2001) descrive SPASS, uno dei più potenti dimostratori di teoremi disponibili attualmente. Il dimostratore di teoremi che ha ottenuto maggiori successi nelle recenti competizioni annuali è stato VAMPIRE (Riazanov e Voronkov, 2002). Anche il sistema COQ (Bertot *et al.*, 2004) e il risolutore di equazioni E (Schulz, 2004) si sono dimostrati validi strumenti per provare la correttezza.

I dimostratori di teoremi sono stati utilizzati per la sintesi automatica e per la verifica di software per il controllo di veicoli spaziali (Denney *et al.*, 2006), tra cui la nuova capsula Orion della NASA (Lowry, 2008). La correttezza del progetto del microprocessore a 32 bit FM9001 è stata provata dal sistema NQTHM (Hunt e Brock, 1992).

La Conference on Automated Deduction (CADE) tiene ogni anno una gara per dimostratori automatici di teoremi. Sutcliffe (2016) descrive la gara del 2016. Tra i sistemi con i migliori risultati vi sono VAMPIRE (Riazanov e Voronkov, 2002), PROVER9 (Sabri, 2015) e una versione aggiornata di E (Schulz, 2013). Wiedijk (2003) mette a confronto la potenza di 15 dimostratori matematici. TPTP (Thousands of Problems for Theorem Provers) è una libreria di problemi di dimostrazione di teoremi, utile per confrontare le prestazioni dei sistemi (Sutcliffe e Suttner, 1998; Sutcliffe *et al.*, 2006).

I dimostratori di teoremi hanno consentito di ottenere nuovi risultati matematici che gli studiosi non erano riusciti a raggiungere in decenni, come è spiegato dettagliatamente nel libro *Automated Reasoning*

and the Discovery of Missing Elegant Proofs (Wos e Pieper, 2003). Il programma SAM (Semi-Automated Mathematics) è stato il primo, dimostrando un lemma della teoria dei reticolati (Guard *et al.*, 1969). Anche il programma AURA ha risposto a questioni aperte in diversi campi della matematica (Wos e Winker, 1983). Il dimostratore di teoremi di Boyer-Moore (Boyer e Moore, 1979) è stato usato da Natarajan Shankar per fornire una dimostrazione formale del teorema di incompletezza di Gödel (Shankar, 1986). Il sistema NUPRL ha dimostrato il paradosso di Girard (Howe, 1987) e il lemma di Higman (Murthy e Russell, 1990).

Nel 1933 Herbert Robbins propose un semplice insieme di assiomi, l'**algebra di Robbins**, che sembrava definire l'algebra booleana, ma non fu trovata una dimostrazione (nonostante il serio impegno di Alfred Tarski e altri) finché EQP (una versione di OTTER) ne trovò una (McCune, 1997). Benzmüller e Paleo (2013) usarono un dimostratore di teoremi di ordine superiore per verificare la prova matematica dell'esistenza di "Dio" di Gödel. Il teorema dell'impacchettamento delle sfere di Kepler fu dimostrato da Thomas Hales

(2005) con l'aiuto di complessi calcoli svolti al computer, ma la dimostrazione fu accettata del tutto soltanto quando fu generata una dimostrazione formale con l'aiuto degli assistenti dimostratori HOL Light e Isabelle (Hales *et al.*, 2017).

Molti vecchi articoli di logica matematica sono raccolti in *From Frege to Gödel: A Source Book in Mathematical Logic* (van Heijenoort, 1967). Tra i libri di testo orientati alla deduzione automatica vi è il classico *Symbolic Logic and Mechanical Theorem Proving* (Chang e Lee, 1973) oltre a lavori più recenti di Wos *et al.* (1992), Bibel (1993) e Kaufmann *et al.* (2000). La rivista più importante nel campo della dimostrazione di teoremi è il *Journal of Automated Reasoning*; i congressi principali sono l'annuale Conference on Automated Deduction (CADE) e l'International Joint Conference on Automated Reasoning (IJCAR). *Handbook of Automated Reasoning* (Robinson e Voronkov, 2001) raccoglie articoli sulla disciplina. *Mathematizing Proof* di MacKenzie (2004) tratta la storia e la tecnologia della dimostrazione di teoremi rivolgendosi al grande pubblico.

CAPITOLO

10

Rappresentazione della conoscenza

- 10.1 Ingegneria ontologica
- 10.2 Categorie e oggetti
- 10.3 Eventi
- 10.4 Oggetti mentali e logica modale
- 10.5 Sistemi di ragionamento per categorie
- 10.6 Ragionare con informazioni di default
- 10.7 Riepilogo
 - Note storiche e bibliografiche

In cui mostriamo come rappresentare diversi aspetti del mondo reale in una forma che possa essere usata per ragionare e per risolvere problemi.

Nei capitoli precedenti abbiamo mostrato come un agente con una base di conoscenza possa effettuare inferenze che gli consentono di agire in modo appropriato. In questo capitolo affronteremo il problema di *quale contenuto* inserire nella base di conoscenza degli agenti: ovvero, come rappresentare fatti che riguardano il mondo. Utilizzeremo la logica del primo ordine come linguaggio di rappresentazione, mentre in capitoli successivi introdurremo diversi formalismi di rappresentazione quali le reti gerarchiche per la pianificazione (Capitolo 11), le reti bayesiane per il ragionamento in condizioni di incertezza (Capitolo 13), i modelli di Markov per il ragionamento nel tempo (Capitolo 17) e le reti neurali deep per il ragionamento su immagini, suoni e altri dati (Capitolo 21 del Volume 2). A prescindere dalla rappresentazione utilizzata, tuttavia, occorre gestire i fatti riguardo il mondo, e questo capitolo vi aiuterà a capire come.

Il Paragrafo 10.1 introduce l'idea di un'ontologia generale, che organizza tutti gli elementi del mondo in una gerarchia di categorie. Il Paragrafo 10.2 tratta le categorie base degli oggetti, delle sostanze e delle misure. Il Paragrafo 10.3 tratta gli eventi e il Paragrafo 10.4 esamina la conoscenza riguardante le credenze. Torniamo poi a considerare la tecnologia per il ragionamento: il Paragrafo 10.5 esamina i sistemi di ragionamento progettati per un'inferenza efficiente con le categorie, mentre il Paragrafo 10.6 discute il ragionamento con informazioni di default.

ingegneria
ontologica

ontologia superiore

10.1 Ingegneria ontologica

Nei domini “giocattolo”, la scelta della rappresentazione non è molto importante; molte scelte funzionano. D’altra parte, domini complessi come gli acquisti su Internet, o la guida di un’automobile nel traffico, richiedono rappresentazioni più generali e flessibili. Questo capitolo mostra come creare tali rappresentazioni concentrandosi su concetti generali che si presentano in molti domini diversi, come *Eventi*, *Tempo*, *Oggetti Fisici* e *Credenze*. Rappresentare questi concetti astratti prende talvolta il nome di **ingegneria ontologica**.

Non possiamo sperare di rappresentare *l’intero* mondo, sarebbe troppo anche per un libro di 1000 pagine: lasceremo invece dei “segnaposto” per indicare i punti in cui si potrà aggiungere nuova conoscenza appartenente a qualsiasi dominio. Definiremo per esempio che cosa significa essere un oggetto fisico, mentre i dettagli dei diversi tipi di oggetto (robot, televisori, libri, o qualsiasi altra cosa) potranno essere aggiunti in seguito. È una modalità analoga a quella con cui i progettisti di framework per la programmazione orientata agli oggetti (come Java Swing, per la grafica) definiscono concetti generali come *Finestra* nella previsione che gli utenti se ne serviranno per definire concetti più specifici come *FinestraFoglioElettronico*. Il framework generale dei concetti prende il nome di **ontologia superiore**, per la convenzione di disegnare grafi con i concetti più generali in alto e quelli più specifici sotto di essi, come si vede nella Figura 10.1.

Prima di approfondire ulteriormente la nozione di ontologia occorre chiarire un punto importante. Abbiamo stabilito di usare la logica del primo ordine per discutere il contenuto e l’organizzazione della conoscenza, ma alcuni aspetti del mondo reale non sono facili da catturare con la logica dei predicati. La difficoltà principale sta nel fatto che la maggior parte delle generalizzazioni prevedono delle eccezioni, o sono vere fino a un certo punto. Per esempio, la regola “i pomodori sono rossi” in genere è utile, ma esistono pomodori gialli, verdi o arancioni. Eccezioni simili possono essere trovate per quasi tutte le regole in questo capitolo. La capacità di gestire eccezioni e incertezza è estremamente importante, ma del tutto ortogonale alla comprensione di un’ontologia generale. Per questa ragione rimanderemo la discussione delle eccezioni al Paragrafo 10.5 e l’argomento più generale dell’informazione incerta al Capitolo 12.

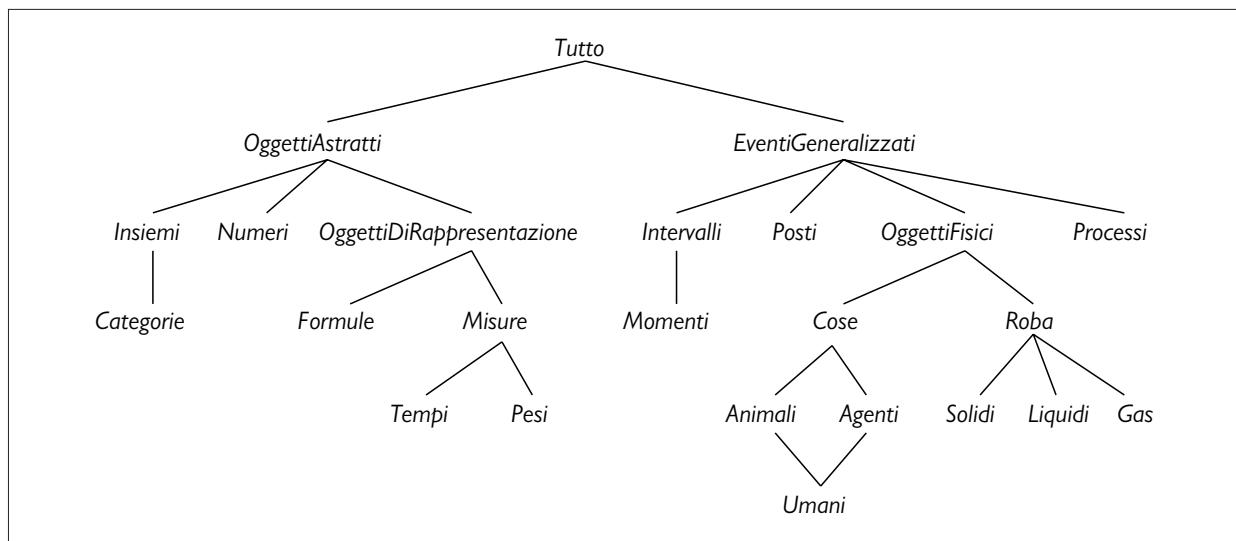


Figura 10.1 L’ontologia superiore del mondo, con gli argomenti che tratteremo nel capitolo. Ogni collegamento indica che il concetto inferiore è una specializzazione di quello superiore. Le specializzazioni non sono necessariamente disgiunte: un umano è sia un animale sia un agente. Nel Paragrafo 10.3.2 vedremo perché gli oggetti fisici si pongono sotto gli eventi generalizzati.

Qual è l'utilità di un'ontologia superiore? Considerate ancora l'ontologia per i circuiti del Paragrafo 8.4.2 che è costruita con molte semplificazioni: il tempo è totalmente mancante; i segnali sono fissi e non si propagano; la struttura del circuito rimane costante. Un'ontologia più generale considererebbe i segnali in momenti precisi e includerebbe la lunghezza delle piste e i ritardi di propagazione. Questo ci permetterebbe di simulare le proprietà temporali del circuito, cosa che i progettisti hardware fanno spesso. Potremmo anche introdurre altre classi interessanti di porte logiche, per esempio descrivendone la tecnologia (TTL, CMOS e così via) oltre che specificandone gli input/output. Se volessimo discutere l'affidabilità o la diagnostica degli errori, dovremmo includere la possibilità che la struttura del circuito o le proprietà delle porte logiche possano cambiare spontaneamente. Per trattare correttamente le capacità parassite, dovremmo rappresentare i luoghi in cui le connessioni si trovano sulla scheda.

Se guardiamo il mondo del wumpus possiamo fare considerazioni simili. Sebbene il tempo sia incluso nella rappresentazione, la sua struttura è semplice: non accade nulla se non quando l'agente esegue un'azione, e tutti i cambiamenti hanno durata istantanea. Un'ontologia più generale, e più adatta al mondo reale, dovrebbe prevedere la possibilità che si verifichino cambiamenti simultanei estesi nel tempo. Abbiamo anche usato un predicato *Pozzo* per indicare quali stanze contenevano pozzi: avremmo potuto rappresentarne tipi diversi definendo più individui appartenenti alla classe dei pozzi, ognuno con differenti caratteristiche. In modo analogo, potremmo desiderare di includere altri animali oltre al wumpus. Potrebbe anche non essere possibile dedurre la specie esatta dalle percezioni disponibili, nel qual caso dovremmo costruire una tassonomia biologica del mondo del wumpus per aiutare l'agente a orientarsi con pochi indizi.

A ogni ontologia specializzata è possibile apportare modifiche come queste per aumentarne la generalità. Sorge quindi spontanea la domanda: tutte queste ontologie convergono in un'unica ontologia generale? Dopo secoli di pensiero filosofico e computazionale, la risposta è: “Può darsi”. In questo paragrafo presentiamo un'ontologia di carattere generale che rappresenta la sintesi delle idee che si sono susseguite nei secoli. Le ontologie generali hanno due caratteristiche che le distinguono da semplici collezioni di ontologie dedicate.

- Un'ontologia generale, con l'aggiunta di assiomi specifici, dovrebbe essere applicabile a quasi tutti i domini particolari. Questo significa che nessun problema di rappresentazione può essere risolto con soluzioni *ad hoc* o ignorato come se non esistesse.
- In qualsiasi dominio sufficientemente complicato occorre *unificare* aree diverse di conoscenza, perché il ragionamento e la risoluzione di problemi possono coinvolgere più aree contemporaneamente. Un sistema robotico di riparazione di circuiti, per esempio, deve essere in grado di ragionare sui circuiti in termini di collegamenti elettrici e configurazione spaziale, ma anche sul tempo, sia per analizzare i circuiti stessi che per stimare il costo delle riparazioni. Deve quindi essere possibile combinare le formule che descrivono il tempo con quelle che descrivono configurazioni spaziali, e le formule devono applicarsi ugualmente bene a nanosecondi e minuti, angstrom e metri.

È giusto dire subito che l'ingegneria ontologica finora ha ottenuto un successo limitato. Nessuna delle più importanti applicazioni dell'IA (elencate nel Capitolo 1) utilizza un'ontologia generale, tutte utilizzano applicazioni specifiche dell'ingegneria della conoscenza e dell'apprendimento automatico. Considerazioni sociali e politiche possono ostacolare il raggiungimento di un accordo su un'ontologia tra parti in competizione. Come dice Tom Gruber (2004), “Ogni ontologia è un patto, un accordo sociale tra persone che hanno un interesse comune verso la condivisione”. Quando gli interessi contrapposti pesano più della spinta alla condivisione, non può esserci un'ontologia comune. Più piccolo è il numero di stakeholder, più facile è creare un'ontologia, per questo è più difficile creare un'ontologia generale di una specifica per uno scopo particolare, come l'Open Biomedical Ontology (Smith *et al.*, 2017). Le ontologie esistenti sono state create seguendo quattro strade:

1. Grazie all’azione di un team di ontologi o logici specializzati, che architettano l’ontologia e scrivono assiomi. Il sistema CYC è stato realizzato in gran parte così (Lenat e Guha, 1990).
2. Importando categorie, attributi e valori da uno o più database esistenti. DBPEDIA è stato costruito importando fatti strutturati da Wikipedia (Bizer *et al.*, 2007).
3. Analizzano documenti di testo ed estraendone informazioni. TEXTRUNNER è stato realizzato leggendo un ampio corpus di pagine web (Banko ed Etzioni, 2008).
4. Convincendo appassionati privi di competenze particolari a inserire conoscenze di senso comune. Il sistema OPENMIND è stato realizzato da volontari che hanno proposto fatti in inglese (Singh *et al.*, 2002; Chklovski e Gil, 2005).

A titolo di esempio, la base di conoscenza Google Knowledge Graph utilizza contenuti semistrutturati di Wikipedia combinandoli con altri contenuti ottenuti dal Web con un lavoro controllato da esseri umani. Contiene oltre 70 miliardi di fatti e fornisce risposte a circa un terzo delle ricerche svolte su Google (Dong *et al.*, 2014).

10.2 Categorie e oggetti

categoria



L’organizzazione degli oggetti in **categorie** è una parte fondamentale della rappresentazione della conoscenza. Benché l’interazione con il mondo abbia luogo a livello di singoli oggetti, *molta parte del ragionamento si svolge al livello delle categorie*. Per esempio, un cliente potrebbe avere come obiettivo l’acquisto di un pallone da basket, piuttosto che di un *particolare* pallone come il BB₉. Le categorie sono anche utili per formulare predizioni sugli oggetti una volta che sono stati classificati. Si può inferire la presenza di un oggetto dagli input percettivi, dedurre dalle caratteristiche percepite la sua appartenenza a una categoria e quindi sfruttare l’informazione nota sulla categoria per formulare predizioni riguardo l’oggetto. Per esempio se è verde e giallo, ha la buccia maculata, ha un diametro di trenta centimetri, ha forma vagamente ovale, polpa rossa, semi neri e sta nelle corsie della frutta si può inferire che l’oggetto è un’anguria e da questo dedurre che non starebbe male in una macedonia.

Per rappresentare le categorie nella logica del primo ordine ci sono due possibilità: si possono usare predicati oppure oggetti. Questo significa che possiamo usare il predicato *PalloneDaBasket(b)*, oppure possiamo **reificare**¹ la categoria e farla diventare un oggetto, *PalloniDaBasket*. A quel punto potremmo scrivere *Membro(b, PalloniDaBasket)*, abbreviato in *b ∈ PalloniDaBasket*, per affermare che *b* è un membro della categoria dei palloni da basket. Possiamo anche scrivere *Sottoinsieme(PalloniDaBasket, Palloni)*, abbreviato in *PalloniDaBasket ⊂ Palloni*, per dire che *PalloniDaBasket* è una **sottocategoria** di *Palloni*. Utilizzeremo in modo intercambiabile i termini sottocategoria, sottoclasse e sottoinsieme.

Le categorie organizzano la conoscenza attraverso il meccanismo dell’**ereditarietà**. Se diciamo che tutte le istanze della categoria *Cibo* sono commestibili, e affermiamo che *Frutta* è una sottoclasse di *Cibo* e *Mele* è a sua volta una sottoclasse di *Frutta*, sappiamo che ogni mela è commestibile. Diciamo che le singole mele **ereditano** la proprietà della commestibilità, in questo caso grazie alla loro appartenenza alla categoria *Cibo*.

La relazione di sottoclasse organizza le categorie in una **gerarchia tassonomica**, o **tassonomia**. Le tassonomie sono state usate esplicitamente per secoli in varie discipline scientifiche.

reificazione

sottocategoria

ereditarietà

gerarchia tassonomica

¹ La trasformazione di una proposizione in un oggetto si dice **reificazione**, dal termine latino *res*, che significa “cosa”. John McCarthy propose il termine “thingification”, traducibile più o meno in “cosificazione”, che tuttavia non fu mai adottato.

che; la più grande di questo tipo organizza circa 10 milioni di specie viventi ed estinte, molte delle quali di scarafaggi,² in una singola gerarchia; la biblioteconomia ne ha sviluppata una di tutte le branche del sapere, rappresentata con il sistema decimale Dewey; il fisco e gli altri ministeri governativi hanno dato origine a complesse tassonomie delle varie occupazioni e dei prodotti commerciali. Le tassonomie sono anche un aspetto importante del buon senso comune.

La logica del primo ordine rende facile esprimere fatti che riguardano intere categorie, mettendo gli oggetti in relazione con esse o quantificando tutti i membri. Presentiamo di seguito alcuni fatti di esempio.

- Un oggetto è membro di una categoria. Per esempio:

$$BB_9 \in PalloniDaBasket.$$

- Una categoria è una sottoclasse di un'altra categoria. Per esempio:

$$PalloniDaBasket \subset Palloni.$$

- Tutti i membri di una categoria hanno determinate caratteristiche. Per esempio:

$$x \in PalloniDaBasket \Rightarrow Sferico(x).$$

- I membri di una categoria possono essere riconosciuti dalle loro caratteristiche. Per esempio:

$$Arancione(x) \wedge Rotondo(x) \wedge Diametro(x)=24\text{cm} \wedge x \in Palloni \Rightarrow x \in PalloniDaBasket.$$

- Una categoria può avere caratteristiche proprie. Per esempio:

$$Cani \in SpecieAddomesticate.$$

Notate che, dato che *Cani* è una categoria ed è membro di *SpecieAddomesticate*, quest'ultima dev'essere una categoria di categorie. Naturalmente esistono anche eccezioni per molte delle precedenti regole (i palloni da basket forati non sono sferici); ne parleremo più avanti.

Benché le relazioni di sottoclasse e di appartenenza siano le più importanti, è anche possibile esprimere relazioni tra categorie che non sono sottoclassi l'una dell'altra. Per esempio, se ci limitiamo a dire che *StudentiTriennale* e *StudentiMagistrale* sono sottoclassi di *StudentiUniversitari*, non abbiamo espresso il fatto che uno studente di un corso di laurea triennale non può essere anche uno studente di un corso di laurea magistrale. Se due categorie non hanno membri in comune, si dice che sono **disgiunte**. Potremmo anche voler dire che le classi di studenti per laurea triennale e per laurea magistrale formano una **scomposizione esaustiva** degli studenti universitari (non considerando gli studenti di dottorato e quelli delle lauree a ciclo unico). Una scomposizione esaustiva disgiunta prende il nome di **partizione**. Riportiamo alcuni esempi di questi tre concetti:

categorie disgiunte
scomposizione
esaustiva
partizione

$$Disgiunte(\{Animali, Vegetali\})$$

$$ScomposizioneEsaustiva(\{Americani, Canadesi, Messicani\}, NordAmericaneri)$$

$$Partizione(\{Animali, Piante, Funghi, Protisti, Monere\}, EsseriViventi).$$

Notate che la scomposizione esaustiva dei NordAmericaneri non è una partizione, perché alcune persone hanno una doppia cittadinanza. I tre predicati sono definiti come segue:

$$Disgiunte(s) \Leftrightarrow (\forall c_1, c_2 \quad c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow Intersezione(c_1, c_2) = \{\})$$

$$ScomposizioneEsaustiva(s, c) \Leftrightarrow (\forall i \quad i \in c \Leftrightarrow \exists c_2 \quad c_2 \in s \wedge i \in c_2)$$

$$Partizione(s, c) \Leftrightarrow Disgiunte(s) \wedge ScomposizioneEsaustiva(s, c).$$

² Quando gli fu chiesto che cosa si potesse dedurre riguardo il Creatore dallo studio della natura, Il famoso biologo J. B. S. Haldane disse: “Una smodata passione per gli scarafaggi”.

Le categorie possono anche essere *definite* fornendo condizioni necessarie e sufficienti per l'appartenenza. Per esempio, uno scapolo è un maschio adulto non sposato:

$$x \in \text{Scapoli} \Leftrightarrow \text{NonSposato}(x) \wedge x \in \text{Adulti} \wedge x \in \text{Maschi}.$$

Come vedremo nel riquadro dedicato ai tipi naturali, una definizione logica precisa delle categorie solitamente è possibile soltanto per termini formali artificiali, non per oggetti del mondo reale. E, del resto, le definizioni non sono sempre necessarie.

10.2.1 Composizione fisica

L'idea che un oggetto possa essere parte di un altro oggetto è abbastanza intuitiva: il nostro naso fa parte della nostra faccia, la Romania fa parte dell'Europa, e questo capitolo fa parte di un libro. Useremo la relazione generale *ParteDi* per affermare che una cosa fa parte di un'altra. Gli oggetti possono essere raggruppati in gerarchie *ParteDi*, che ricordano le gerarchie di sottoinsiemi:

$$\begin{aligned} &\text{ParteDi(Bucarest, Romania)} \\ &\text{ParteDi(Romania, EuropaOrientale)} \\ &\text{ParteDi(EuropaOrientale, Europa)} \\ &\text{ParteDi(Europa, Terra).} \end{aligned}$$

La relazione *ParteDi* è transitiva e riflessiva:

$$\begin{aligned} &\text{ParteDi}(x, y) \wedge \text{ParteDi}(y, z) \Rightarrow \text{ParteDi}(x, z) \\ &\text{ParteDi}(x, x). \end{aligned}$$

Possiamo quindi concludere che vale *ParteDi(Bucarest, Terra)*.

oggetto composto

Le categorie di **oggetti composti** sono spesso caratterizzate da relazioni strutturali tra le parti. Per esempio, un bipede è un oggetto con esattamente due gambe attaccate a un corpo:

$$\begin{aligned} \text{Bipede}(a) \Rightarrow & \exists l_1, l_2, b \text{ Gamba}(l_1) \wedge \text{Gamba}(l_2) \wedge \text{Corpo}(b) \wedge \\ & \text{ParteDi}(l_1, a) \wedge \text{ParteDi}(l_2, a) \wedge \text{ParteDi}(b, a) \wedge \\ & \text{Attaccata}(l_1, b) \wedge \text{Attaccata}(l_2, b) \wedge \\ & l_1 \neq l_2 \wedge [\forall l_3 \text{ Gamba}(l_3) \wedge \text{ParteDi}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)]. \end{aligned}$$

La notazione per esprimere “esattamente due” è un po' arzigogolata; siamo obbligati a dire che esistono due gambe, che non sono lo stesso oggetto e che se qualcuno fa riferimento a una terza gamba, essa deve corrispondere a una delle prime due. Nel Paragrafo 10.5.2 presentiamo il formalismo della logica descrittiva, che rende molto più facile esprimere vincoli come questo.

Possiamo definire una relazione *PartizioneDiParti* con un significato analogo a quello di *Partizione* per le categorie (cfr. Esercizio 10.DECM). Un oggetto è composto dalle parti che costituiscono la sua *PartizioneDiParti* e può anche derivare da esse alcune caratteristiche: per esempio, la massa di un oggetto composto è la somma delle masse delle sue parti. Notate che non è affatto così con le categorie, che non hanno alcuna massa, anche se i loro elementi potrebbero averla.

Può anche essere utile definire oggetti composti con parti definite ma nessuna particolare struttura. Potremmo per esempio voler dire che “le mele in questo sacchetto pesano due chili”. Potremmo essere tentati di assegnare questo peso all'*insieme* delle mele nel sacchetto, ma questo sarebbe un errore, perché un insieme è un concetto matematico astratto che può contenere elementi ma non può avere alcun peso. Occorre un nuovo concetto, che chiameremo **mucchio** (*bunch*). Per esempio, se le mele sono *Mela*₁, *Mela*₂ e *Mela*₃, allora

$$\text{MucchioDi}(\{\text{Mela}_1, \text{Mela}_2, \text{Mela}_3\})$$

denota l'oggetto composto che ha le tre mele come parti (e non come elementi). A questo punto sarà possibile usare il mucchio come un oggetto normale, ancorché privo di struttura. Notate che $MucchioDi(\{x\}) = x$. Inoltre, $MucchioDi(Mele)$ è l'oggetto composto da tutte le mele esistenti, e non dev'essere confuso con $Mele$, la categoria o insieme di tutte le mele.

Possiamo definire $MucchioDi$ partendo dalla relazione $ParteDi$. È ovvio che ogni elemento di s fa parte di $MucchioDi(s)$:

$$\forall x \ x \in s \Rightarrow ParteDi(x, MucchioDi(s)).$$

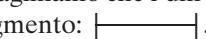
Inoltre, $MucchioDi(s)$ è il più piccolo oggetto che soddisfa questa condizione. In altre parole, $MucchioDi(s)$ dev'essere parte di ogni oggetto che ha come parti tutti gli elementi di s :

$$\forall y \ [\forall x \ x \in s \Rightarrow ParteDi(x, y)] \Rightarrow ParteDi(MucchioDi(s), y).$$

Questi assiomi sono un esempio della tecnica generale chiamata **minimizzazione logica**, che permette di definire un oggetto come il più piccolo che soddisfa certe condizioni.

minimizzazione logica

10.2.2 Misure

Sia per la scienza che per il buon senso, gli oggetti hanno un'altezza, una massa, un costo e così via. I valori che assegniamo a queste caratteristiche prendono il nome di **misure**. Le misure quantitative ordinarie sono abbastanza facili da rappresentare. Immaginiamo che l'universo includa "oggetti misura" astratti, come la *lunghezza* di questo segmento:  Possiamo chiamare questa lunghezza 0,53 pollici o 1,35 centimetri. Così, nel nostro linguaggio la stessa lunghezza può avere nomi differenti. Rappresentiamo la lunghezza con una **funzione unità** che accetta come argomento un numero (l'Esercizio 10.ALTM prende in esame uno schema alternativo). Se il nome del segmento è L_1 , possiamo scrivere:

$$Lunghezza(L_1) = Pollici(0,53) = Centimetri(1,35).$$

La conversione tra unità di misura si ottiene esprimendo l'uguaglianza tra multipli di unità diverse:

$$Centimetri(2,54 \times d) = Pollici(d).$$

Assiomi simili possono essere scritti per libbre e chilogrammi, secondi e giorni, dollari e centesimi. Le misure possono essere usate per descrivere oggetti nel modo seguente:

$$Diametro(PalloneDaBasket_{12}) = Pollici(9,5)$$

$$Prezzo(PalloneDaBasket_{12}) = \$19$$

$$Peso(MucchioDi(\{Mela_1, Mela_2, Mela_3\})) = Chilogrammi(2)$$

$$d \in Giorni \Rightarrow Durata(d) = Ore(24)$$

misura

funzione unità

Notate che $\$(1)$ non è un biglietto da un dollaro, ma un prezzo. Si possono possedere due o più banconote uguali, ma c'è un solo oggetto di nome $\$(1)$. Notate anche che, sebbene $Pollici(0)$ e $Centimetri(0)$ si riferiscano alla stessa lunghezza nulla, non sono affatto identici ad altre misure zero, come $Secondi(0)$.

Semplici misure quantitative sono facili da rappresentare; la cosa si fa più difficile quando non esiste una scala di valori precisa. Gli esercizi hanno un grado di difficoltà, i dessert un grado di dolcezza e le poesie di bellezza, tuttavia è impossibile assegnare a queste caratteristiche un valore numerico. Un fanatico della contabilità potrebbe pensare che proprietà come queste non siano utili per il ragionamento logico o, ancora peggio, cercare di imporre una scala numerica per la bellezza. Questo sarebbe un grave errore, perché non è necessario: l'aspetto più importante delle misure infatti non è il loro particolare valore numerico, ma il fatto che possono essere *ordinate*.

Benché le misure non siano numeri, possiamo ancora confrontarle usando un simbolo di ordinamento come $>$. Per esempio, si potrebbe affermare che gli esercizi di Norvig sono più

TIPI NATURALI

Alcune categorie hanno una definizione precisa; un oggetto è un triangolo se e solo se è un poligono con tre lati. D'altra parte, la maggior parte delle categorie del mondo reale non si possono definire esattamente: le chiameremo **tipi naturali**. I pomodori, ad esempio, di solito hanno un colore rosso spento; sono più o meno sferici con una depressione in alto in corrispondenza del picciolo; sono grandi dai cinque ai dieci centimetri; hanno una buccia sottile ma resistente e contengono polpa, succo e semi. Esistono però delle variazioni: alcuni pomodori sono arancioni, quelli acerbi sono verdi, ce ne sono di più grandi o più piccoli della media, e poi ci sono i "cilegini" che sono tutti molto più piccoli. Invece di una definizione precisa dei pomodori, abbiamo una serie di caratteristiche che possono aiutarci a identificare oggetti che sono chiaramente pomodori tipici, ma che potrebbero risultare inadeguate a identificare altri oggetti (potrebbe esistere un pomodoro coperto di peluria, come una pesca?).

Tutto questo per un agente logico rappresenta un problema. L'agente non può essere sicuro che l'oggetto che ha percepito è un pomodoro, e anche se lo fosse, non potrebbe sapere con certezza quali proprietà dei pomodori tipici abbia questa particolare istanza. Questo problema rappresenta l'inevitabile conseguenza dell'operare in un ambiente parzialmente osservabile.

Un approccio utile consiste nel separare ciò che è vero per tutte le istanze di una categoria da ciò che vale solo per le istanze tipiche. Così, oltre alla categoria *Pomodori*, avremo anche la categoria *Tipici(Pomodori)*. La funzione *Tipici* fa corrispondere una categoria alla sua sottoclassificazione che contiene solo istanze tipiche:

$$\text{Tipici}(c) \subseteq c .$$

La maggior parte della conoscenza relativa ai tipi naturali si applicherà alle loro istanze tipiche:

$$x \in \text{Tipici}(\text{Pomodori}) \Rightarrow \text{Rosso}(x) \wedge \text{Sferico}(x) .$$

In questo modo è possibile scrivere informazioni utili sulle categorie senza vincolarsi a definizioni esatte.

La difficoltà di fornire definizioni precise della maggior parte delle categorie naturali è stata analizzata approfonditamente da Wittgenstein (1953), che sfruttò l'esempio dei giochi per mostrare che i membri di una categoria condividono "somiglianze di famiglia" e non caratteristiche necessarie e sufficienti: quale definizione stretta comprende scacchi, wrestling, solitario e palla prigioniera?

La stessa utilità del concetto di definizione precisa fu contestata da Quine (1953), che fece notare che persino la definizione di "scapolo" come maschio adulto non sposato è sospetta. Consideriamo ad esempio una frase come "il Papa è scapolo": sebbene non sia *falsa* in senso stretto, in questo caso l'uso del termine è certamente *infelice*, perché porta a inferenze non volute da parte dell'ascoltatore. Questa tensione potrebbe forse essere risolta distinguendo tra le definizioni logiche, adatte per la rappresentazione della conoscenza interna, e i criteri più sfumati richiesti dall'uso appropriato del linguaggio. Quest'ultimo potrebbe essere ottenuto "filtrando" le asserzioni ricavate dalle definizioni logiche. Potrebbe anche darsi che i fallimenti nell'uso del linguaggio possano servire da feedback per la modifica delle definizioni interne, in modo tale che il filtraggio diventi superfluo.

difficili di quelli di Russell, e che uno studente ottiene meno punti quando deve svolgere esercizi difficili:

$$\begin{aligned} e_1 \in \text{Esercizi} \wedge e_2 \in \text{Esercizi} \wedge \text{ScrittoDa}(\text{Norvig}, e_1) \wedge \text{ScrittoDa}(\text{Russell}, e_2) \Rightarrow \\ \text{Difficoltà}(e_1) > \text{Difficoltà}(e_2) \\ e_1 \in \text{Esercizi} \wedge e_2 \in \text{Esercizi} \wedge \text{Difficoltà}(e_1) > \text{Difficoltà}(e_2) \Rightarrow \\ \text{PunteggioAtteso}(e_1) < \text{PunteggioAtteso}(e_2) . \end{aligned}$$

Questo è sufficiente per permettere a uno studente di decidere quali esercizi svolgere, anche se da nessuna parte è stata usata una scala numerica per la difficoltà (naturalmente, sarà ne-

cessario scoprire chi è l'autore di ogni esercizio). Questo tipo di relazioni monotone tra misure forma la base della **fisica qualitativa**, un sottoinsieme dell'IA che studia il ragionamento su sistemi fisici senza ricorrere a equazioni dettagliate e simulazioni numeriche. Accenneremo alla fisica qualitativa nelle note storiche alla fine del capitolo.

10.2.3 Oggetti: cose e roba

Si può pensare che il mondo reale consista di oggetti primitivi (particelle atomiche) e oggetti composti costruiti per aggregazione. Mantenendo il ragionamento al livello di oggetti grandi come mele e automobili, possiamo gestire la complessità inherente alla trattazione individuale di un grande numero di oggetti primitivi. Tuttavia, una porzione significativa di realtà sembra sfuggire a qualsiasi evidente **individuazione**, intendendo con questo termine la divisione in oggetti distinti. Daremo a questa porzione il nome generico di **roba** (*stuff*). Per esempio, supponiamo di avere di fronte a noi un po' di burro e un formichiere. Si può dire che il formichiere è uno, ma non c'è un numero evidente di "oggetti-burro", dato che ogni parte di un oggetto-burro è anch'essa un oggetto-burro, almeno finché non arriviamo a particelle davvero infinitesimali. Questa è la differenza principale tra roba e cose. Se tagliamo a metà un formichiere, non ne otteniamo due (sfortunatamente).

Il linguaggio naturale distingue chiaramente cose e roba: infatti diciamo "un formichiere", ma non possiamo dire "un burro". I linguisti distinguono tra **sostantivi numerabili** (contabili), come i formichieri, i buchi e i teoremi, e **sostantivi non numerabili** (collettivi) come burro, acqua ed energia. Molte ontologie, in competizione tra loro, sostengono di essere in grado di gestire questa distinzione. Qui ne descriveremo solo una; le altre sono citate nelle note storiche.

Per rappresentare la roba nel modo adeguato, cominciamo dalle cose ovvie. La nostra ontologia dovrà prevedere come oggetti, quantomeno, i "pezzi" di roba con i quali possiamo interagire. Potremmo quindi riconoscere il pezzo di burro come quello che abbiamo lasciato sul tavolo la notte scorsa; potremmo prenderlo, pesarlo, venderlo, e così via. In questa accezione, il burro è un oggetto proprio come il formichiere. Chiamiamolo *Burro*₃. Definiamo anche la categoria *Burro*: informalmente i suoi elementi saranno tutte quelle cose di cui si può dire "ehi, è burro", tra cui *Burro*₃. Lasciando da parte le particelle davvero piccole, che per ora trascureremo, ogni parte di un oggetto-burro è anch'essa un oggetto-burro:

$$b \in \text{Burro} \wedge \text{ParteDi}(p, b) \Rightarrow p \in \text{Burro} .$$

Ora possiamo dire che il burro si scioglie a una temperatura intorno ai 30 gradi centigradi:

$$x \in \text{Burro} \Rightarrow \text{PuntoFusione}(x, \text{Centigradi}(30)) .$$

Potremmo continuare e dire che il burro è giallo, meno denso dell'acqua, morbido a temperatura ambiente, molto grasso, e così via. D'altra parte, il burro non ha una dimensione, una forma o un peso particolari. Possiamo definire categorie specializzate di burro come *BurroSalato*, che sarà anch'esso un tipo di roba. Notate che *ChiloDiBurro*, che include come membri tutti gli oggetti-burro che pesano un chilo, non è una *roba*! Se tagliamo a metà un chilo di burro non otteniamo, ahinoi, due chili di burro.

Tutto questo è dovuto al fatto che alcune proprietà degli oggetti sono **intrinseche**: non appartengono all'oggetto in sé, ma alla sostanza di cui è fatto. Quando tagliate un'istanza di *roba* a metà, i due pezzi mantengono le proprietà intrinseche come la densità, il punto di ebollizione, il sapore, il colore, il proprietario e così via. D'altra parte, le proprietà **estrinseche** – peso, lunghezza, forma e così via – non rimangono inalterate dopo una suddivisione. Una categoria di oggetti che include nella sua definizione solo proprietà *intrinseche* è quindi una sostanza, o sostantivo non numerabile; se invece ha *anche una sola* proprietà estrinseca è un sostantivo numerabile. *Roba* e *Cosa* sono le più generali categorie di sostanze e oggetti, rispettivamente.

individuazione
roba

sostantivo
numerabile
sostantivo
non numerabile

intrinseco
estrinseco

10.3 Eventi

Nel Paragrafo 7.7.1 abbiamo discusso le azioni, cioè atti che avvengono, come *Scocca*³, e i fluenti, aspetti del mondo che cambiano, come *HaFreccia*⁴. Entrambi i concetti sono stati rappresentati come proposizioni, e abbiamo usato assiomi di stato successore per affermare che un fluente sarà vero al tempo $t+1$ se l'azione al tempo t lo avrà fatto diventare tale, o se era già vero al tempo t e l'azione non l'ha fatto diventare falso. Tutto ciò valeva in un mondo in cui le azioni sono discrete, istantanee, si verificano una alla volta e non presentano variazioni nel modo in cui sono eseguite (questo significa che esiste un solo tipo di azione *Scocca*, senza distinzione tra scoccare rapidamente, lentamente, nervosamente e così via).

Quando passiamo da domini semplificati al mondo reale, tuttavia, dobbiamo affrontare una varietà molto più ampia di azioni o eventi.³ Considerate un'azione continua, quale il riempimento di una vasca da bagno. Un assioma di stato successore può affermare che la vasca è vuota prima dell'azione e piena quando l'azione è terminata, ma non può dire nulla di ciò che accade *durante* l'azione. Non può nemmeno descrivere facilmente due azioni che si verificano nello stesso tempo, come il lavarsi i denti mentre si aspetta che la vasca si riempia. Per gestire casi simili introduciamo un approccio noto come **calcolo degli eventi**.

Gli oggetti del calcolo degli eventi sono eventi, fluenti e istanti (punti) temporali. *Posizione(Shankar, Berkeley)* è un fluente, cioè un oggetto che si riferisce al fatto che Shankar si trova a Berkeley. L'evento E_1 di Shankar che vola da San Francisco a Washington, D.C. è descritto come:

$$E_1 \in Voli \wedge Viaggiatore(E_1, Shankar) \wedge Origine(E_1, SF) \wedge Destinazione(E_1, DC).$$

Dove *Voli* è la categoria di tutti gli eventi di volo. Reificando gli eventi rendiamo possibile aggiungere qualsiasi quantità di informazioni arbitrarie su di essi. Per esempio, possiamo dire che il volo di *Shankar* è stato affetto da turbolenze con *Turbolento*(E_1). In un'ontologia in cui gli eventi sono predicati n -ari, non sarebbe possibile aggiungere informazioni extra come queste, e passare a un predicato $n+1$ -ario non è una soluzione scalabile.

Per asserire che un fluente è effettivamente vero a partire da un istante temporale t_1 e lo è rimasto con continuità fino all'istante t_2 utilizziamo il predicato *T*, come in *T(Posizione(Shankar, Berkeley), t₁, t₂)*. In modo simile, utilizziamo *Accade*(E_1, t_1, t_2) per dire che l'evento E_1 si è effettivamente verificato a partire dal tempo t_1 e fino al tempo t_2 . L'insieme completo dei predicati per una versione del calcolo degli eventi⁴ è:

$T(f, t_1, t_2)$	Il fluente f è vero in tutti gli istanti di tempo da t_1 a t_2
$Accade(e, t_1, t_2)$	L'evento e inizia al tempo t_1 e termina al tempo t_2
$Inizia(e, f, t)$	L'evento e fa diventare vero il fluente f al tempo t
$Termina(e, f, t)$	L'evento e fa diventare non più vero il fluente f al tempo t
$Iniziato(f, t_1, t_2)$	Il fluente f è diventato vero in un istante temporale compreso tra t_1 e t_2
$Terminato(f, t_1, t_2)$	Il fluente f ha cessato di essere vero in un istante temporale compreso tra t_1 e t_2
$t_1 < t_2$	L'istante temporale t_1 si presenta prima di t_2

Possiamo descrivere gli effetti di un evento di volo:

$$\begin{aligned} E = Voli(a, qui, là) \text{ e } Accade(E, t_1, t_2) \Rightarrow \\ Termina(E, In(a, qui), t_1) \wedge Inizia(E, In(a, qui), t_2) \end{aligned}$$

³ I termini “evento” e “azione” possono essere utilizzati in modo intercambiabile nel senso di “qualcosa che può accadere”.

⁴ La nostra versione si basa su Shanahan (1999) ma con alcune modifiche.

Ipotizziamo un evento distinto, *Partenza*, che descrive lo stato iniziale affermando quali fluenti sono veri (usando *Inizia*) o falsi (usando *Terminato*) al tempo di partenza. Possiamo poi descrivere quali fluenti sono veri in quali istanti temporali mediante una coppia di assiomi per T e $\neg T$ con lo stesso formato generale degli assiomi di stato successore: si assume che un evento accada tra il tempo t_1 e t_3 e che in un istante t_2 compreso tra i precedenti l'evento modifichi il valore del fluente f , iniziandolo (rendendolo vero) o terminandolo (rendendolo falso). Poi al tempo t_4 nel futuro, se non sono intervenuti eventi a cambiare il fluente (terminandolo o iniziandolo, rispettivamente), allora questo avrà mantenuto il suo valore.

Formalmente gli assiomi sono:

$$\text{Accade}(e, t_1, t_3) \wedge \text{Inizia}(e, f, t_2) \wedge \neg \text{Terminato}(f, t_2, t_4) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \Rightarrow T(f, t_2, t_4)$$

$$\text{Accade}(e, t_1, t_3) \wedge \text{Termina}(e, f, t_2) \wedge \neg \text{Iniziato}(f, t_2, t_4) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \Rightarrow \neg T(f, t_2, t_4)$$

dove *Terminato* e *Iniziato* sono definiti da:

$$\text{Terminato}(f, t_1, t_5) \Leftrightarrow$$

$$\exists e, t_2, t_3, t_4 \text{ Accade}(e, t_2, t_4) \wedge \text{Termina}(e, f, t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

$$\text{Iniziato}(f, t_1, t_5) \Leftrightarrow$$

$$\exists e, t_2, t_3, t_4 \text{ Accade}(e, t_2, t_4) \wedge \text{Inizia}(e, f, t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

Possiamo estendere il calcolo degli eventi per rappresentare eventi simultanei (come quando sono necessarie due persone per far andare un'altalena), eventi esogeni (come il vento che fa muovere un oggetto), eventi continui (come il montare della marea), eventi non deterministici (come il lancio di una moneta a testa o croce) e altre complicazioni.

10.3.1 Tempo

Il calcolo degli eventi ci apre la possibilità di parlare di tempo e di istanti o punti temporali. Considereremo due tipi di intervalli temporali: i momenti e gli intervalli estesi. La distinzione sta nel fatto che i momenti hanno una durata pari a zero:

$$\text{Partizione}(\{\text{Momenti}, \text{IntervalliEstesi}\}, \text{Intervalli})$$

$$i \in \text{Momenti} \Leftrightarrow \text{Durata}(i) = \text{Secondi}(0).$$

Per avere tempi assoluti è necessario inventare una scala temporale e associare i punti su di essa ai momenti. La scala è arbitraria; misureremo i tempi in secondi e stabiliremo che l'istante iniziale, di tempo 0, è la mezzanotte (GMT) dell'1 gennaio 1900. Le funzioni *Inizio* e *Fine* indicano rispettivamente il primo e l'ultimo momento di un intervallo, e la funzione *Tempo* restituisce il riferimento sulla scala temporale assoluta di un dato momento. La funzione *Durata* rappresenta la differenza tra i tempi di fine e di inizio.

$$\text{Intervallo}(i) \Rightarrow \text{Durata}(i) = (\text{Tempo}(\text{Fine}(i)) - \text{Tempo}(\text{Inizio}(i))).$$

$$\text{Tempo}(\text{Inizio}(AD1900)) = \text{Secondi}(0).$$

$$\text{Tempo}(\text{Inizio}(AD2001)) = \text{Secondi}(3187324800).$$

$$\text{Tempo}(\text{Fine}(AD2001)) = \text{Secondi}(3218860800).$$

$$\text{Durata}(AD2001) = \text{Secondi}(31536000).$$

Per aumentare la leggibilità di questi valori possiamo introdurre una funzione *Data*, che prende sei argomenti (ore, minuti, secondi, giorno, mese e anno) e restituisce un punto sulla scala temporale:

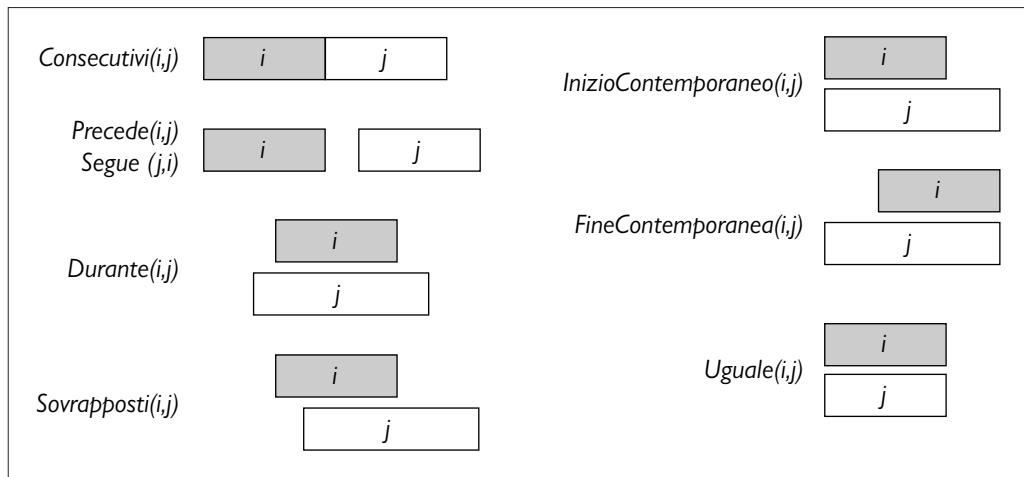
$$\text{Tempo}(\text{Inizio}(AD2001)) = \text{Data}(0, 0, 0, 1, \text{gen}, 2001)$$

$$\text{Data}(0, 20, 21, 24, 1, 1995) = \text{Secondi}(3000000000).$$

Due intervalli sono *Consecutivi* se la fine del primo coincide con l'inizio del secondo. L'insieme completo di relazioni tra intervalli (Allen, 1983) è illustrato di seguito e nella Figura 10.2:

Figura 10.2

Predicati su intervalli temporali.



$$\begin{aligned}
 \text{Consecutivi}(i, j) &\Leftrightarrow \text{Fine}(i) = \text{Inizio}(j) \\
 \text{Precede}(i, j) &\Leftrightarrow \text{Fine}(i) < \text{Inizio}(j) \\
 \text{Segue}(j, i) &\Leftrightarrow \text{Precede}(i, j) \\
 \text{Durante}(i, j) &\Leftrightarrow \text{Inizio}(j) < \text{Inizio}(i) < \text{Fine}(i) < \text{Fine}(j) \\
 \text{Sovrapposti}(i, j) &\Leftrightarrow \text{Inizio}(i) < \text{Inizio}(j) < \text{Fine}(i) < \text{Fine}(j) \\
 \text{InizioContemporaneo}(i, j) &\Leftrightarrow \text{Inizio}(i) = \text{Inizio}(j) \\
 \text{FineContemporanea}(i, j) &\Leftrightarrow \text{Fine}(i) = \text{Fine}(j) \\
 \text{Uguale}(i, j) &\Leftrightarrow \text{Inizio}(i) = \text{Inizio}(j) \wedge \text{Fine}(i) = \text{Fine}(j).
 \end{aligned}$$

In tutti i casi il significato è intuitivo, con l'eccezione di *Sovrapposti*: tendiamo a pensare che la sovrapposizione sia simmetrica (se *i* è sovrapposto a *j* allora *j* è sovrapposto a *i*), ma in questa definizione *Sovrapposti*(*i,j*) è vero solo se *i* inizia prima di *j*. L'esperienza mostra che questa definizione è più utile per la scrittura di assiomi. Per dire che il regno di Elisabetta II ha seguito immediatamente quello di Giorgio VI, e quello di Elvis si è sovrapposto con gli anni 1950, possiamo scrivere quanto segue:

$$\begin{aligned}
 &\text{Consecutivi}(\text{RegnoDi(GiorgioVI)}, \text{RegnoDi}(ElisabettaII)) \\
 &\text{Sovrapposti}(\text{Anni50}, \text{RegnoDi}(Elvis)) \\
 &\text{Inizio}(\text{Anni50}) = \text{Inizio}(AD1950) \\
 &\text{Fine}(\text{Anni50}) = \text{Fine}(AD1959).
 \end{aligned}$$

10.3.2 Fluenti e oggetti

Gli oggetti fisici possono essere considerati eventi generalizzati, nel senso che un oggetto fisico è anch'esso un pezzo di spazio-tempo. Gli Stati Uniti, per esempio, possono essere considerati un evento cominciato nel 1776 come unione di 13 stati e ancora in atto oggi come unione di 50.

Possiamo descrivere le proprietà in evoluzione di *StatiUniti* per mezzo dei fluenti di stato, come *Popolazione(StatiUniti)*. Una proprietà degli Stati Uniti che, salvo imprevisti, cambia ogni quattro od ogni otto anni è il suo presidente. Si potrebbe pensare che il termine logico *Presidente(StatiUniti)* possa indicare oggetti diversi in tempi differenti: sfortunatamente non è così, dato che in un modello un termine denota esattamente un oggetto. Il termine *Presidente(StatiUniti, t)* può riferirsi a oggetti diversi a seconda del valore di *t*, ma la nostra ontologia tiene separati indici temporali e fluenti. L'unica possibilità è che *Presidente(StatiUniti)* denoti effettivamente un singolo oggetto, ma che questo consista di persone diverse in tempi

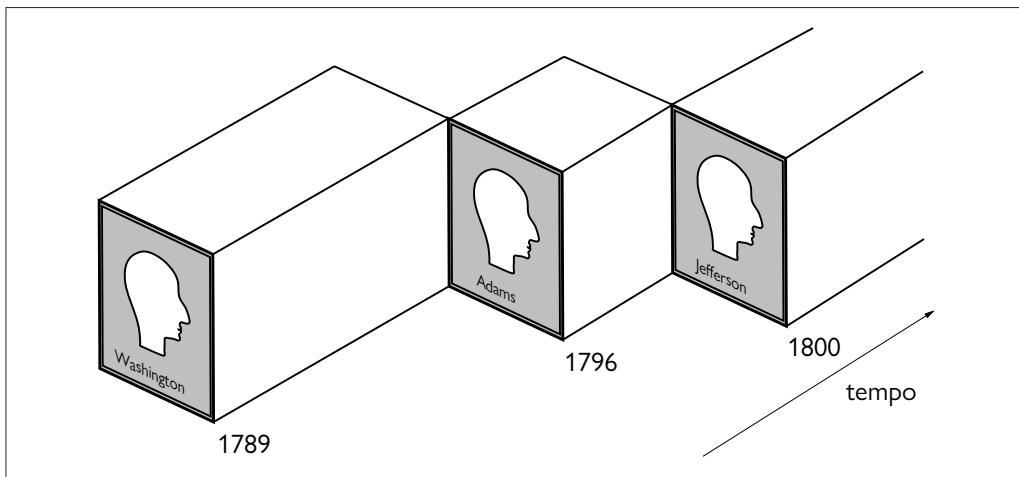


Figura 10.3
Una vista schematica dell'oggetto *Presidente(Stati Uniti)* per i primi anni.

diversi: sarà l'oggetto che è George Washington dal 1789 al 1797, John Adams dal 1797 al 1801 e così via, come si vede nella Figura 10.3. Per dire che George Washington è stato presidente per tutto il 1790, possiamo scrivere:

$$T(Uguale(\text{Presidente}(\text{StatiUniti}), \text{GeorgeWashington}), \text{Inizio}(AD1790), \text{Fine}(AD1790)).$$

Utilizziamo il simbolo di funzione *Uguale* anziché il predicato logico standard = perché non possiamo avere un predicato come argomento di *T*, e perché l'interpretazione *non* è che *GeorgeWashington* e *Presidente(StatiUniti)* sono logicamente identici nel 1790; l'identità logica del resto non è qualcosa che può cambiare con il tempo. L'identità è tra i sottoeventi degli oggetti *Presidente(StatiUniti)* e *GeorgeWashington* identificati dal periodo 1790.

10.4 Oggetti mentali e logica modale

Gli agenti che abbiamo costruito fin qui hanno credenze, e ne possono dedurre di nuove. Tuttavia, nessuno di essi possiede conoscenza *che riguarda* le stesse credenze o lo stesso processo di deduzione. La conoscenza riguardante la propria stessa conoscenza e i propri processi di ragionamento può essere utile per controllare l'inferenza. Per esempio, supponiamo che Alice chieda: “Qual è la radice quadrata di 1764” e Bruno risponda: “Non lo so”. Se Alice insiste con “Pensaci meglio”, Bruno dovrebbe rendersi conto che, pensandoci ancora un po’, in effetti è possibile rispondere alla domanda. D’altra parte, se la domanda fosse: “Il presidente è seduto in questo momento?”, Bruno dovrebbe rendersi conto che anche pensandoci di più non risolverebbe molto. Anche la conoscenza sulla conoscenza di altri agenti è importante: Bruno dovrebbe rendersi conto che il presidente sa se è seduto o no, e chiedere a lui sarebbe un modo per rispondere alla domanda.

Quello che ci serve è un modello degli oggetti mentali nella testa delle persone (o nella base di conoscenza degli agenti) e dei processi mentali che manipolano tali oggetti. Il modello non deve essere dettagliato. Non occorre prevedere quanti millisecondi serviranno a un agente per dedurre un fatto. Ci basterà essere in grado di concludere che il presidente sa se è seduto o no.

Iniziamo con le **attitudini proposizionali** che un agente può avere verso gli oggetti mentali: attitudini quali *Crede*, *Conosce*, *Vuole* e *Informa*. La difficoltà è data dal fatto che queste attitudini non si comportano come predici “normali”. Per esempio, supponiamo di provare ad asserire che Lois sa che Superman può volare:

$$\text{Conosce}(Lois, \text{PuòVolare}(Superman)).$$

attitudine proposizionale

Un piccolo problema è che normalmente consideriamo *PuòVolare(Superman)* come una formula, ma qui appare come termine. Questo problema si può risolvere reificando *PuòVolare(Superman)*; facendone un fluente. Un problema più grave è che, se è vero che Superman è Clark Kent, allora dobbiamo concludere che Lois sa che Clark può volare, sbagliando perché (nella maggior parte delle versioni della storia) Lois *non* sa che Clark è Superman:

$$\begin{aligned} (\text{Superman} = \text{Clark}) \wedge \text{Conosce}(\text{Lois}, \text{PuòVolare}(\text{Superman})) \\ \models \text{Conosce}(\text{Lois}, \text{PuòVolare}(\text{Clark})). \end{aligned}$$

Questo è una conseguenza del fatto che il ragionamento sull'uguaglianza è integrato nella logica. Normalmente è una buona cosa; se il nostro agente sa che $2 + 2 = 4$ e $4 < 5$, allora vogliamo che sappia che $2 + 2 < 5$. Questa proprietà si chiama **trasparenza referenziale**: non conta quale termine è utilizzato da una logica per fare riferimento a un oggetto, conta l'oggetto a cui il termine dà il nome. Tuttavia, per attitudini proposizionali come *crede* e *conosce*, preferiremmo avere l'opacità referenziale: i termini utilizzati contano, perché non tutti gli agenti sanno quali termini sono coreferenziali.

Potremmo aggiungere un'altra reificazione: avere un oggetto che rappresenti Clark/Superman, un altro che rappresenti la persona che Lois conosce come Clark e un altro ancora che rappresenti la persona che Lois conosce come Superman. Tuttavia, questa proliferazione di oggetti farebbe diventare le formule lunghe e poco eleganti.

trasparenza referenziale

logica modale

mondi possibili
relazioni di
accessibilità

La **logica modale** è pensata per risolvere questo problema. La logica normale si occupa di una sola modalità, quella della verità, che ci consente di esprimere “*P* è vero” o “*P* è falso”. La logica modale include speciali operatori modali che accettano come argomenti formule (e non termini). Per esempio, “*A* conosce *P*” è rappresentato con la notazione $\mathbf{K}_A P$, dove \mathbf{K} è l'operatore modale per la conoscenza. Questo operatore richiede due argomenti, un agente (scritto come pedice) e una formula. La sintassi della logica modale è identica a quella della logica del primo ordine, tranne per il fatto che le formule possono anche essere formate con operatori modali.

La semantica della logica modale è più complessa. Nella logica del primo ordine un **modello** contiene un insieme di oggetti e un'interpretazione che associa ogni nome all'oggetto, relazione o funzione appropriati. In logica modale vogliamo poter considerare la possibilità che l'identità segreta di Superman sia Clark e la possibilità che non lo sia.

Ci serve quindi un modello più complesso, costituito da una collezione di **mondi possibili**, anziché da un unico mondo vero. I mondi sono connessi in un grafo mediante **relazioni di accessibilità**, una per ogni operatore modale. Diciamo che il mondo w_1 è accessibile dal mondo w_0 rispetto all'operatore modale \mathbf{K}_A se in w_1 tutto è consistente con ciò che *A* conosce in w_0 . Per esempio, nel mondo reale Bucarest è la capitale della Romania, ma per un agente che non lo sa, un mondo in cui la capitale della Romania fosse Sofia sarebbe accessibile. Si spera che un mondo in cui $2 + 2 = 5$ non sia accessibile ad alcun agente.

In generale, un atomo di conoscenza $\mathbf{K}_A P$ è vero nel mondo w se e solo se *P* è vero in ogni mondo accessibile da w . La verità di formule più complesse è derivata per applicazione ricorsiva di questa regola e delle regole normali della logica del primo ordine. Ciò significa che la logica modale può essere utilizzata per ragionare su formule di conoscenza annidate: ciò che un agente sa della conoscenza di un altro agente. Per esempio possiamo dire che, anche se Lois non sa se l'identità segreta di Superman è Clark Kent, sa che Clark lo sa:

$$\mathbf{K}_{\text{Lois}} [\mathbf{K}_{\text{Clark}} \text{Identità}(\text{Superman}, \text{Clark}) \vee \mathbf{K}_{\text{Clark}} \neg \text{Identità}(\text{Superman}, \text{Clark})].$$

La logica modale risolve alcuni problemi difficili facendo entrare in gioco quantificatori e conoscenza. La frase in italiano: “Bond sa che qualcuno è una spia” è ambigua. La prima lettura è che esiste un “qualcuno” in particolare che Bond sa essere una spia; possiamo scrivere:

$$\exists x \mathbf{K}_{\text{Bond}} \text{Spia}(x),$$

che in logica modale significa che esiste un x che, in tutti i mondi accessibili, Bond sa essere una spia. La seconda lettura è che Bond sa soltanto che esiste almeno una spia:

$$\mathbf{K}_{\text{Bond}} \exists x \text{ Spia}(x).$$

L'interpretazione in logica modale è che in ogni mondo accessibile esiste un x che è una spia, ma non è necessariamente lo stesso x in ogni mondo.

Ora che abbiamo un operatore modale per la conoscenza, possiamo scrivere assiomi per esso. In primo luogo, possiamo dire che gli agenti sono in grado di trarre conclusioni; se un agente conosce P e conosce che P implica Q , allora l'agente P conosce Q :

$$(\mathbf{K}_a P \wedge \mathbf{K}_a(P \Rightarrow Q)) \Rightarrow \mathbf{K}_a Q.$$

Da questo (e da altre regole sulle identità logiche) possiamo stabilire che $\mathbf{K}_A(P \vee \neg P)$ è una tautologia; ogni agente sa che ogni proposizione P è vera o falsa. D'altra parte, $(\mathbf{K}_A P) \vee (\mathbf{K}_A \neg P)$ non è una tautologia; in generale, ci saranno molte proposizioni per cui un agente non sa essere vere e non sa essere false.

È stato detto (risalendo a Platone) che la conoscenza è credenza vera giustificata. Se è vero, se lo credete, e se avete una ragione inoppugnabile, allora lo conoscete. Ciò significa che, se conoscete qualcosa, deve essere vera, e abbiamo l'assioma:

$$\mathbf{K}_a P \Rightarrow P.$$

Inoltre, gli agenti logici (ma non tutte le persone) sono capaci di introspezione sulla loro conoscenza. Se conoscono qualcosa, allora conoscono che lo conoscono:

$$\mathbf{K}_a P \Rightarrow \mathbf{K}_a(\mathbf{K}_a P).$$

Possiamo definire assiomi simili per la credenza (spesso denotata da **B**) e altre modalità. Tuttavia, un problema dell'approccio basato sulla logica modale è che assume una **onniscienza logica** da parte degli agenti. Ovvero, se un agente conosce un insieme di assiomi, allora conosce tutte le conseguenze di tali assiomi. Tutto ciò costituisce un terreno malfermo anche per la nozione in qualche modo astratta di conoscenza, ma sembra ancora peggio per la credenza, che ha una maggiore connotazione di riferimento a cose che sono fisicamente rappresentate nell'agente, non solo potenzialmente derivabili.

Ci sono stati tentativi di definire una forma di razionalità limitata per gli agenti, di dire che gli agenti credono solo le asserzioni che possono essere derivate applicando non più di k passi di ragionamento, o non più di s secondi di elaborazione, ma in generale tali tentativi sono risultati insoddisfacenti.

10.4.1 Altre logiche modali

Sono state proposte molte logiche modali per diverse modalità, oltre alla conoscenza. Una proposta è quella di aggiungere operatori modali per i concetti di *possibilità* e *necessità*: è possibile che sia vero che uno degli autori di questo libro sia seduto in questo momento, ed è necessariamente vero che $2+2 = 4$.

Come si è detto nel Paragrafo 8.1.2, alcuni studiosi di logica preferiscono le modalità riferite al tempo. Nella **logica temporale lineare** si aggiungono i seguenti operatori modali:

- **X** P : “ P sarà vero al prossimo passo temporale”
- **F** P : “ P sarà finalmente vero in qualche passo temporale futuro”
- **G** P : “ P è sempre (globalmente) vero”
- **P** **U** Q : “ P rimane vero fino a che Q diventa vero”

A volte ci sono altri operatori che si possono derivare da questi. L'aggiunta di questi operatori modali rende la logica più complessa (e quindi aumenta la difficoltà di trovare una di-

mostrazione per un algoritmo di inferenza logica), ma d'altra parte ci consente di affermare certi fatti in una forma più breve (il che rende più rapida l'inferenza logica). Per scegliere la logica da usare si procede come per la scelta di un linguaggio di programmazione: si sceglie quella più appropriata al compito da svolgere, che sia familiare a chi la deve usare e agli altri che condivideranno il lavoro, ed efficiente quanto basta per lo scopo prefissato.

10.5 Sistemi di ragionamento per categorie

reti semantiche

logiche descrittive

grafo esistenziale

Le categorie sono gli elementi principali per la costruzione di schemi di rappresentazione della conoscenza su grande scala. Questo paragrafo descrive i sistemi esplicitamente progettati per organizzare e ragionare sulle categorie. Ci sono due famiglie di sistemi, strettamente imparentate: le **reti semantiche** permettono di visualizzare graficamente una base di conoscenza e forniscono algoritmi efficienti per inferire le proprietà di un oggetto sulla base della sua appartenenza a una categoria; le **logiche descrittive** rappresentano un linguaggio formale per costruire e combinare definizioni di categorie e forniscono algoritmi efficienti per determinare le relazioni di sottoinsieme e sovrainsieme tra categorie.

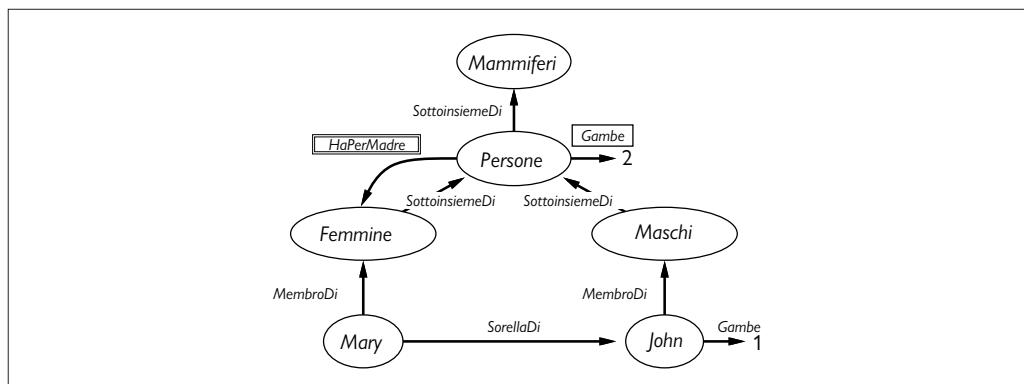
10.5.1 Reti semantiche

Nel 1909 Charles Peirce propose una notazione grafica basata su nodi e archi che chiamò **grafi esistenziali** e definì “la logica del futuro”. Cominciò così un lungo dibattito tra i paladini della “logica” e quelli delle “reti semantiche”. Sfortunatamente, le diatribe nascosero il fatto che le reti semantiche *solo* sono una forma di logica. Per certi tipi di formule la notazione offerta dalle reti semantiche è spesso più comoda, ma se non consideriamo le questioni di “interfacciamento” con gli esseri umani, i concetti sottostanti – oggetti, relazioni, quantificazioni e così via – sono identici.

Esistono molte varianti di reti semantiche, ma tutte sono capaci di rappresentare oggetti singoli, categorie di oggetti e relazioni fra gli oggetti. Una notazione grafica tipica rappresenta i nomi degli oggetti e delle categorie racchiusi in ovali o rettangoli, connessi con collegamenti etichettati. Per esempio, nella Figura 10.4 c'è un collegamento *MembroDi* tra *Mary* e *Femmine*, che corrisponde all'asserzione logica $Mary \in Femmine$; analogamente, il collegamento *SorellaDi* tra *Mary* e *John* corrisponde all'asserzione *SorellaDi(Mary, John)*. Possiamo mettere in relazione categorie mediante collegamenti *SottoinsiemeDi*, e così via. Disegnare bolle e frecce è così divertente che ci si può anche lasciar prendere la mano. Per esempio, sappiamo che tutte le persone hanno come madre una femmina; è possibile quindi tracciare un collegamento *HaPerMadre* da *Persone* a *Femmine*? La risposta è no, perché *HaPerMadre* è una

Figura 10.4

Una rete semantica con quattro oggetti (John, Mary, 1, 2) e quattro categorie. Le relazioni sono rappresentate da archi etichettati.



relazione tra una persona e sua madre, mentre le categorie non hanno madri.⁵ Per questa ragione, nella Figura 10.4 abbiamo usato come notazione speciale un collegamento la cui etichetta è racchiusa da un doppio rettangolo. Tale collegamento asserisce che:

$$\forall x \ x \in \text{Persone} \Rightarrow [\forall y \ \text{HaPerMadre}(x, y) \Rightarrow y \in \text{Femmine}] .$$

Potremmo anche voler asserire che le persone hanno due gambe, ovvero:

$$\forall x \ x \in \text{Persone} \Rightarrow \text{Gambe}(x, 2) .$$

Anche in questo caso dobbiamo stare attenti a non dire che una categoria ha le gambe; il collegamento nella Figura 10.4 ha un'etichetta racchiusa in un rettangolo singolo ed è usato per asserire le proprietà comuni a ogni membro di una categoria.

La notazione delle reti semantiche rende molto semplice eseguire ragionamenti sull'**ereditarietà** del tipo che abbiamo introdotto nel Paragrafo 10.2. In virtù del fatto di essere una persona, per esempio, Mary eredita la proprietà di avere due gambe. Così, per determinare il numero di gambe di Mary, l'algoritmo di ereditarietà seguirà il collegamento *MembroDi* da *Mary* alla categoria a cui ella appartiene e da lì seguirà la catena di collegamenti *SottoinsiemeDi* finché non troverà una categoria che ha un collegamento etichettato dalla scritta *Gambe* racchiusa in un rettangolo: in questo caso, *Persone*. La semplicità e l'efficienza di questo meccanismo di inferenza rispetto alla dimostrazione logica di teoremi è stata una delle attrattive principali delle reti semantiche.

L'ereditarietà diventa complicata quando un oggetto può appartenere a più di una categoria, o quando una categoria può essere un sottoinsieme di più di un'altra categoria; in questo caso si parla di **ereditarietà multipla**. In questi casi l'algoritmo potrebbe trovare, in risposta a una query, due o più valori in conflitto. Per questa ragione alcuni linguaggi di **programmazione orientata agli oggetti** (OOP), come Java, proibiscono l'uso dell'ereditarietà multipla nella gerarchia delle classi. Nelle reti semantiche, comunque, essa è generalmente consentita: ne ripareremo nel Paragrafo 10.6.

**ereditarietà
multipla**

Il lettore avrà già notato un'ovvia limitazione delle reti semantiche rispetto alla logica del primo ordine, e cioè il fatto che i collegamenti tra le bolle possono rappresentare solo relazioni *binarie*. La formula *Vola(Shankar, NewYork, NuovaDelhi, Ieri)*, per esempio, non può essere espressa direttamente in una rete semantica. Questo non toglie che sia possibile ottenere l'effetto di un'asserzione *n*-aria; per far questo si deve reificare la proposizione stessa come un evento appartenente a una categoria di eventi appropriata. La Figura 10.5 mostra

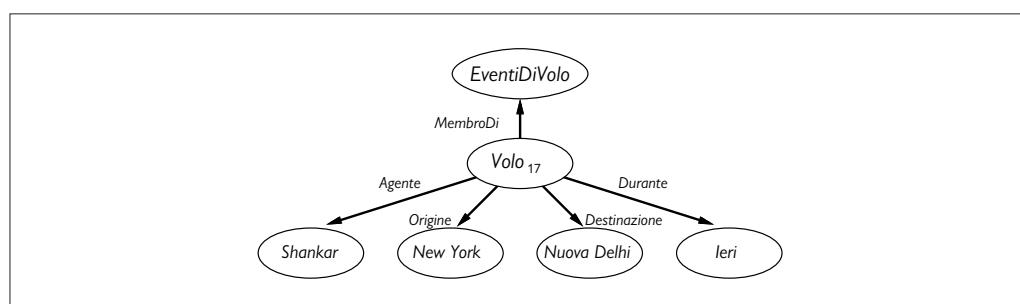


Figura 10.5
Un frammento di rete semantica che mostra la rappresentazione dell'asserzione logica *Vola(Shankar, NewYork, NuovaDelhi, Ieri)*.

⁵ Molti dei primi sistemi non riuscivano a distinguere tra le proprietà dei membri di una categoria e quelle della categoria stessa. Questo può portare a delle gravi inconsistenze, come ha fatto notare Drew McDermott (1976) nel suo articolo "Artificial Intelligence Meets Natural Stupidity". Un altro problema comune era l'uso di collegamenti *IsA* (è un) per indicare sia la relazione di sottoinsieme che quella di appartenenza a una categoria, come nel linguaggio naturale: "un gatto è un mammifero" e "Cirillo è un gatto". L'Esercizio 10.NATS esplora ulteriormente quest'argomento.

la struttura di una rete semantica per questo particolare evento. Notate che la restrizione delle relazioni binarie ha come conseguenza la creazione di una ricca ontologia di concetti reificati.

La reificazione delle proposizioni rende possibile rappresentare ogni formula atomica, ground e priva di funzioni della logica del primo ordine con la notazione delle reti. Alcuni tipi di formule universalmente quantificate possono essere espresse usando collegamenti inversi e con frecce etichettate con rettangoli singoli e doppi applicati alle categorie; questo tuttavia ci lascia ben lontani dalla potenza espressiva della logica del primo ordine completa. Tra le altre cose non possiamo esprimere la negazione, la disgiunzione, funzioni annidate e la quantificazione esistenziale. È *possibile* estendere la notazione per renderla del tutto equivalente alla logica del primo ordine, come per gli stessi grafi esistenziali di Peirce, questo però elimina uno dei vantaggi principali delle reti semantiche: la semplicità e la trasparenza dei processi di inferenza. I progettisti possono costruire una grande rete e continuare ad avere una buona idea di quali interrogazioni risulteranno efficienti, perché (a) è facile visualizzare i passi della procedura di inferenza e (b) in alcuni casi il linguaggio di query è così semplice che non si possono proprio esprimere interrogazioni complesse. Nei casi in cui la potenza espressiva è troppo limitata, per colmare le lacune molte reti semantiche sfruttano il cosiddetto meccanismo del **collegamento procedurale** (*procedural attachment*): questa tecnica fa sì che una query (e talvolta anche un'asserzione) che riguarda una certa relazione abbia come risultato la chiamata di una procedura speciale appositamente progettata, senza utilizzare l'algoritmo generale di inferenza.

collegamento procedurale

valore di default

sovrascrittura

Uno degli aspetti più importanti delle reti semantiche è la capacità di rappresentare **valori di default** per le categorie. Esaminando attentamente la Figura 10.4, potete notare che John ha una sola gamba, nonostante il fatto che sia una persona e che tutte le persone abbiano due gambe. In una KB strettamente logica, questa sarebbe una contraddizione; in una rete semantica, tuttavia, l'asserzione che tutte le persone hanno due gambe ha solo un valore di default. Questo significa che si presume sempre che le persone abbiano due gambe a meno che ciò non sia contraddetto da informazione più specifica. La semantica dei valori di default è supportata in modo naturale dall'algoritmo di ereditarietà, perché quest'ultimo segue i link verso l'alto partendo dall'oggetto (John, in questo caso) e fermandosi non appena trova un valore. Si dice che il valore di default è **sovrascritto** (*overridden*) dal valore più specifico. Notate che potremmo anche sovrascrivere il numero di gambe di default creando una categoria di *PersoneConUnaGamba*, un sottoinsieme di *Persone* di cui *John* sarebbe membro.

Per mantenere la stretta correttezza logica della rete, possiamo dire che l'asserzione *Gambe* delle *Persone* include un'eccezione nel caso di John:

$$\forall x \ x \in \text{Persone} \wedge x \neq \text{John} \Rightarrow \text{Gambe}(x, 2).$$

Per una rete di struttura *prefissata* questo è semanticamente accettabile, ma nel caso ci siano molte eccezioni le espressioni logiche saranno molto meno concise della notazione grafica. Nel caso che la rete debba essere aggiornata con nuove asservizioni, poi, quest'approccio fallirà completamente: dovremmo dire che anche tutte le persone (presentemente sconosciute) con una sola gamba costituiscono un'eccezione alla regola. Il Paragrafo 10.6 approfondisce questo problema e il ragionamento di default in generale.

10.5.2 Logiche descrittive

logica descrittiva

La sintassi della logica del primo ordine è progettata per rendere facile parlare degli oggetti. Le **logiche descrittive** sono notazioni progettate per facilitare la descrizione delle definizioni e delle proprietà delle categorie. I sistemi basati sulla logica descrittiva si sono evoluti dalle reti semantiche per rispondere alla necessità di formalizzare il significato delle reti, mantenendo l'enfasi sulla struttura tassonomica come principio organizzativo.

```

Concetto → Thing | NomeDiConcetto
| And(Concetto, . . .)
| All(NomeDiRuolo, Concetto)
| AtLeast(Intero, NomeDiRuolo)
| AtMost(Intero, NomeDiRuolo)
| Fills(NomeDiRuolo, NomeIndividuale, . . .)
| SameAs(Cammino, Cammino)
| OneOf(NomeIndividuale, . . .)
Cammino → [NomeDiRuolo, . . .]
NomeDiConcetto → Adulato | Femmina | Maschio | . . .
NomeDiRuolo → Coniuge | Figlia | Figlio | . . .

```

Figura 10.6 La sintassi delle descrizioni in un sottoinsieme del linguaggio CLASSIC.

Il principale compito inferenziale delle logiche descrittive è la **sussunzione** (che consiste nel determinare se una categoria è un sottoinsieme di un'altra confrontando le loro definizioni) e la **classificazione** (la verifica che un oggetto appartiene a una categoria). Alcuni sistemi includono anche la verifica di **consistenza** di una definizione di categoria: una definizione è consistente se i criteri di appartenenza sono logicamente soddisfacibili.

sussunzione
classificazione
consistenza

Il linguaggio CLASSIC (Borgida *et al.*, 1989) è una tipica logica descrittiva. La sintassi delle descrizioni CLASSIC è mostrata nella Figura 10.6.⁶ Per esempio, per dire che gli scapoli sono maschi adulti non sposati scriveremo:

Scapolo = And (NonSposato, Adulato, Maschio).

In logica del primo ordine, la formula equivalente sarebbe:

Scapolo (x) ⇔ NonSposato(x) ∧ Adulato(x) ∧ Maschio(x).

Notate che la logica descrittiva ha un'algebra di operazioni sui predicati, cosa che naturalmente non si può fare in logica del primo ordine. Ogni descrizione in CLASSIC può essere tradotta in una formula equivalente in logica del primo ordine, ma in alcuni casi CLASSIC è più semplice. Per esempio, per descrivere l'insieme degli uomini con almeno tre figli tutti disoccupati e sposati con dottori e con al più due figlie che sono tutte professoresse di fisica o matematica, potremmo scrivere:

*And(Uomo, AtLeast(3, Figlio), AtMost(2, Figlia),
All(Figlio, And(Disoccupato, Sposato, All(Coniuge, Dottore))),
All(Figlia, And(Professore, Fills(Dipartimento, Fisica, Matematica)))).*

Lasciamo ai lettori l'esercizio di tradurre tutto ciò in logica del primo ordine.

L'aspetto più importante delle logiche descrittive è forse la loro enfasi sulla trattabilità dell'inferenza. Un'istanza di problema è risolta descrivendola e poi chiedendo al sistema se è sussunta da una tra molte possibili categorie di soluzioni. Nei sistemi standard basati sulla logica del primo ordine, predire il tempo richiesto per il calcolo di una soluzione è quasi sempre impossibile. Quando un problema richiede settimane di tempo di calcolo, è spesso compito dell'utente organizzare la rappresentazione in modo da “girare intorno” agli insiemi

⁶ Notate che il linguaggio non permette di affermare semplicemente che un concetto, o una categoria, è il sottoinsieme di un altro. Questa è una scelta voluta: la sussunzione tra categorie dev'essere derivabile da aspetti delle loro stesse descrizioni. Se questo non è il caso, è segno che le descrizioni sono in qualche modo carenti.

di formule che sembrano causare il rallentamento. Uno degli scopi principali della logica descrittiva, al contrario, è assicurare che la verifica di sussunzione possa essere effettuata in un tempo polinomiale rispetto alla dimensione delle descrizioni.⁷

In linea di principio sembra tutto fantastico, finché non si realizza che ci possono essere due sole possibili conseguenze: i problemi difficili non potranno proprio essere espressi, o le loro descrizioni dovranno essere esponenzialmente grandi! I risultati sulla trattabilità del problema, comunque, permettono di riconoscere i costrutti che causano problemi e aiutano quindi l’utente a capire come si comportano le diverse rappresentazioni.

Le logiche descrittive normalmente non sono dotate di *negazione* e *disgiunzione*, che obbligano i sistemi del primo ordine a eseguire un’analisi potenzialmente esponenziale per assicurare la completezza. CLASSIC prevede una forma limitata di disgiunzione nei costrutti *Fills* e *OneOf*, che permettono di esprimere disgiunzioni su elementi esplicitamente elencati ma non sulle descrizioni. Se le descrizioni fossero disgiuntive, definizioni annidate potrebbero facilmente portare a un numero esponenziale di cammini alternativi per cui una categoria può sussuderne un’altra.

10.6 Ragionare con informazione di default

Nel paragrafo precedente abbiamo incontrato un semplice esempio di asserzione con uno stato di default: le persone hanno due gambe. Questo default può essere sovrascritto da informazione più specifica, come il fatto che Long John Silver ha una gamba sola. Come abbiamo visto, il meccanismo dell’ereditarietà nelle reti semantiche implementa la sovrascrittura dei valori di default in modo semplice e naturale. In questo paragrafo studieremo i default in modo più generale, cercando di capire la loro *semantica* e non solo di fornire meccanismi procedurali.

10.6.1 Circoscrizione e logica di default

monotonicità

Abbiamo visto due esempi di processi di ragionamento apparentemente naturali che violavano la proprietà della **monotonicità** della logica che abbiamo dimostrato nel Capitolo 7.⁸ In questo capitolo abbiamo visto che una proprietà ereditata da tutti i membri di una categoria in una rete semantica poteva essere sovrascritta da informazione più specifica relativa a una sottocategoria. Nel Paragrafo 9.4.4 abbiamo visto che, sotto l’ipotesi del mondo chiuso, se una proposizione α non è menzionata in KB , allora $KB \models \neg\alpha$, ma $KB \wedge \alpha \models \alpha$.

non monotonicità

Una semplice introspezione ci suggerisce che queste eccezioni alla monotonicità sono estremamente frequenti nel ragionamento comune: sembra che le persone tendano a “saltare alle conclusioni”. Guardando una macchina parcheggiata, per esempio, tenderemo a credere che abbia quattro ruote anche se ne vediamo solo tre. La teoria della probabilità può aiutarci a concludere che la quarta ruota esiste quasi certamente, ma non è questo il punto: per la maggior parte delle persone, la possibilità che la macchina non abbia quattro ruote *non si pone a meno che non si presenti nuova informazione in suo supporto*. Sembra quindi che la conclusione sul numero delle ruote sia raggiunta *per default*, a meno che non ci siano ragioni per dubitarne. Qualora dovessero presentarsi altre informazioni (se dovesse per esempio vedere che il proprietario sta trasportando una ruota, e la macchina è stata sollevata con il cric) la conclusione potrebbe essere ritrattata. Si dice che questo tipo di ragionamento è **non monotono**, perché l’insieme delle credenze non cresce monotonicamente

⁷ CLASSIC offre un controllo della sussunzione efficiente in pratica, ma il tempo di esecuzione nel caso pessimo è esponenziale.

⁸ Ricordiamo che la monotonicità richiede che tutte le formule che sono conseguenze logiche rimangano tali dopo l’aggiunta di nuove formule alla KB. In altre parole, se $KB \models \alpha$ allora $KB \wedge \beta \models \alpha$.

nel tempo man mano che giungono nuove informazioni. Per catturare questo tipo di comportamento sono state inventate **logiche non monotone**, che prevedono definizioni diverse di verità e conseguenza logica. Esamineremo due logiche che sono state studiate estensivamente: la circoscrizione e la logica di default.

logica non monotona

La **circoscrizione** può essere vista come una versione più precisa e potente dell'ipotesi del mondo chiuso. L'idea è specificare particolari predicati che si presumono "più falsi possibile": in altre parole, saranno falsi per tutti gli oggetti tranne quelli per cui è noto che sono veri. Supponiamo per esempio di voler esprimere la regola di default secondo cui gli uccelli volano. Dovremo introdurre un predicato, diciamo *Anormale*₁(*x*), e scrivere:

$$Uccello(x) \wedge \neg Anormale_1(x) \Rightarrow Vola(x).$$

Se diciamo che *Anormale*₁ dev'essere **circoscritto**, un ragionatore circoscrittivo potrà dare per scontato $\neg Anormale_1(x)$ a meno che la verità di *Anormale*₁(*x*) sia nota. Questo permette di concludere *Vola(Titti)* dalla premessa *Uccello(Titti)*, ma la conclusione non varrà più qualora si appurasse che *Anormale*₁(*Titti*).

circoscritto

La circoscrizione può essere considerata un esempio di logica con **preferenza di modelli**. In tali logiche una formula è conseguenza logica (per default) se è vera in tutti i modelli *preferiti* della KB, mentre nella logica classica il requisito è che sia vera in *tutti* i modelli possibili. Nella circoscrizione un modello è preferito a un altro se ha meno oggetti anormali.⁹ Vediamo come funziona quest'idea nel contesto dell'ereditarietà multipla nelle reti semantiche. L'esempio classico per illustrare i problemi di ereditarietà multipla è chiamato il "rombo di Nixon". Richard Nixon era infatti sia un Quacchero (e quindi per default un pacifista) che un Repubblicano (per default, non pacifista). Possiamo scriverlo come segue:

preferenza di modelli

$$\begin{aligned} & Repubblicano(Nixon) \wedge Quacchero(Nixon) \\ & Repubblicano(x) \wedge \neg Anormale_2(x) \Rightarrow \neg Pacifista(x) \\ & Quacchero(x) \wedge \neg Anormale_3(x) \Rightarrow Pacifista(x). \end{aligned}$$

Se circoscriviamo *Anormale*₂ e *Anormale*₃, ci sono due modelli preferiti: nel primo *Anormale*₂(*Nixon*) e *Pacifista*(*Nixon*) sono veri, nell'altro sono veri *Anormale*₃(*Nixon*) e $\neg Pacifista(Nixon)$. Il ragionatore circoscrittivo, quindi, rimarrà correttamente agnostico riguardo alla natura pacifista di Nixon. Se, oltre a tutto ciò, vogliamo asserire che le credenze religiose hanno la precedenza su quelle politiche, possiamo usare un formalismo chiamato **circoscrizione con priorità** che permette di dare la preferenza ai modelli in cui *Anormale*₃ è minimizzato.

circoscrizione con priorità

La **logica di default** è un formalismo in cui si possono scrivere **regole di default** per generare conclusioni contingenti non monotone. Una regola di default ha quest'aspetto:

logica di default
regola di default

$$Uccello(x) : Vola(x) / Vola(x).$$

Questa regola significa che se *Uccello(x)* è vero, e *Vola(x)* è consistente con la base di conoscenza, allora si può concludere *Vola(x)* per default. In generale, una regola di default ha la forma:

$$P : J_1, \dots, J_n / C$$

dove *P* è il prerequisito, *C* la conclusione e *J_i* le giustificazioni: se una qualsiasi di esse può essere dimostrata falsa, non si può trarre la conclusione. Tutte le variabili che compaiono

⁹ Per l'ipotesi del mondo chiuso, un modello è preferito a un altro se ha meno atomi veri: in altre parole, i modelli preferiti sono quelli **minimi**. C'è un'affinità naturale tra l'ipotesi del mondo chiuso e le KB composte da clausole definite, perché il punto fisso raggiunto dalla concatenazione in avanti su tali KB è il modello minimo unico (cfr. Paragrafo 7.5.4).

in J_i o C devono anche comparire in P . L'esempio del rombo di Nixon può essere rappresentato in logica di default con un fatto e due regole di default:

$$\begin{aligned} &\text{Repubblicano(Nixon)} \wedge \text{Quacchero(Nixon)} \\ &\text{Repubblicano}(x) : \neg \text{Pacifista}(x) / \neg \text{Pacifista}(x) \\ &\text{Quacchero}(x) : \text{Pacifista}(x) / \text{Pacifista}(x). \end{aligned}$$

estensione

Per interpretare il significato delle regole di default definiamo il concetto di **estensione** di una teoria di default come l'insieme massimale di conseguenze di tale teoria. In altre parole, un'estensione S consiste nei fatti noti originali e nell'insieme di conclusioni delle regole di default tale che non sia possibile trarre nuove conclusioni da S e che le giustificazioni di ogni conclusione di default in S siano consistenti con S stesso. Come nel caso dei modelli preferiti nella circoscrizione, abbiamo due possibili estensioni per il rombo di Nixon: in una è pacifista, nell'altra non lo è. Esistono schemi con priorità nei quali alcune regole di default possono avere la precedenza su altre, permettendo così di risolvere qualche ambiguità.

Dal 1980, anno in cui sono state proposte per la prima volta le logiche non monotone, sono stati fatti molti progressi nella comprensione delle loro proprietà matematiche. Tuttavia, rimangono ancora molte questioni irrisolte. Per esempio, se la formula “le macchine hanno quattro ruote” è falsa, che cosa significa averla nella propria base di conoscenza? Qual è un buon insieme di regole di default? Se non possiamo decidere se una regola, presa singolarmente, appartiene o no alla base di conoscenza, sorgono dei seri problemi di non-modularità. Infine, com’è possibile usare credenze di default per prendere decisioni? Questo è probabilmente il problema più spinoso del ragionamento di default. Le decisioni spesso richiedono di valutare dei compromessi, e quindi diventa necessario confrontare la *forza* delle credenze nei risultati di azioni diverse e il *costo* di prendere una decisione sbagliata. Nei casi in cui viene preso ripetutamente lo stesso tipo di decisioni, è possibile interpretare le regole di default come asserzioni di “soglia di probabilità”: per esempio, la regola di default “i miei freni sono sempre a posto” significherà in realtà “la probabilità che i miei freni siano a posto, in assenza di altre informazioni, è sufficientemente alta che la decisione ottima sarà sempre quella di cominciare a guidare senza controllarli”. Quando il contesto della decisione muta – per esempio, quando si sta guidando un autocarro carico lungo una ripida strada di montagna – la regola di default diventa improvvisamente inappropriata, anche se non ci sono nuove informazioni che facciano pensare che i freni siano rotti. Queste considerazioni hanno portato i ricercatori a considerare la possibilità di integrare il ragionamento di default nella teoria della probabilità.

revisione delle credenze

10.6.2 Sistemi di mantenimento della verità

Abbiamo visto che molte delle inferenze prodotte da un sistema di rappresentazione della conoscenza avranno solo uno stato di default, anziché essere assolutamente certe. Inevitabilmente alcuni di questi fatti inferiti si riveleranno errati e, alla luce di ulteriori informazioni, dovranno essere ritrattati. Questo processo prende il nome di **revisione delle credenze**.¹⁰ Supponiamo che una base di conoscenza KB contenga una formula P errata – forse una conclusione di default ottenuta da un processo di concatenazione in avanti, o forse sem-

¹⁰ La revisione delle credenze viene spesso contrapposta all'**aggiornamento delle credenze**, che si verifica quando una base di conoscenza viene modificata per riflettere un cambiamento nel mondo piuttosto che la disponibilità di ulteriori informazioni riguardanti un mondo fisso. L'aggiornamento delle credenze combina la revisione delle credenze con il ragionamento sul tempo e sul cambiamento; è imparentato anche al processo di **filtraggio** che descriveremo nel Capitolo 14.

plicemente un'asserzione scorretta – e di voler eseguire l'azione $\text{TELL}(KB, \neg P)$. Per non creare una contraddizione, si deve prima eseguire $\text{RETRACT}(KB, P)$. Fin qui tutto sembra facile. I problemi sorgono quando formule *aggiuntive* sono state inferite da P e inserite nella KB: per esempio, l'implicazione $P \Rightarrow Q$ potrebbe aver causato l'aggiunta di Q . La soluzione intuitiva di rimuovere anche tutte le formule inferite da P non è praticabile, perché tali formule potrebbero avere altre giustificazioni oltre a P . Riprendendo l'esempio di prima, se nella KB fossero presenti anche R e $R \Rightarrow Q$, allora Q non dovrebbe affatto essere rimossa. I **sistemi di mantenimento della verità**, o TMS (*truth maintenance systems*), sono progettati per gestire proprio questo tipo di complicazioni.

Un approccio molto semplice al mantenimento della verità consiste nel tener traccia dell'ordine con cui le formule sono inserite nella base di conoscenza numerandole da P_1 a P_n . Al momento della chiamata $\text{RETRACT}(KB, P_i)$ il sistema ritorna allo stato immediatamente precedente all'inserimento di P_i , rimuovendo quindi sia P_i che tutte le formule da essa inferite. In seguito, le formule da P_{i+1} a P_n possono essere aggiunte nuovamente. Questa soluzione è semplice, e garantisce la consistenza della base di conoscenza, ma la ritrattazione di P_i richiede che siano ritrattate (e poi asserte nuovamente) $n - i$ formule; quel che è peggio è che dovranno essere annullate e rieseguite anche tutte le inferenze tratte da esse. Per i sistemi con un grande numero di fatti, come le grandi basi di dati commerciali, tutto ciò non è praticabile.

Un approccio più efficiente è rappresentato dai sistemi di mantenimento della verità basati sulla giustificazione, o **JTMS** (*justification-based truth maintenance system*). In un JTMS, ogni formula nella base di conoscenza è annotata con una **giustificazione** che consiste nell'insieme di formule da cui è stata inferita. Per esempio, se la base di conoscenza contiene già $P \Rightarrow Q$, allora $\text{TELL}(KB, P)$ causerà anche l'aggiunta di Q con la giustificazione $\{P, P \Rightarrow Q\}$. In generale, una formula può avere un numero qualsiasi di giustificazioni. Esse servono a rendere efficienti le ritrattazioni: alla chiamata $\text{RETRACT}(KB, P)$, il JTMS cancellerà esattamente le formule in cui P è membro di ogni giustificazione. Così una formula Q con la singola giustificazione $\{P, P \Rightarrow Q\}$ sarà rimossa, con la giustificazione aggiuntiva $\{P, P \vee R \Rightarrow Q\}$ sarà ancora rimossa, ma questo non accadrà nel caso la formula abbia anche la giustificazione $\{R, P \vee R \Rightarrow Q\}$. In questo modo il tempo richiesto per la ritrattazione di P dipende solo dal numero di formule derivate da essa e non da tutte le altre formule eventualmente aggiunte dopo P .

Il JTMS presume sempre che le formule considerate una volta lo saranno probabilmente ancora, così invece di cancellarla fisicamente dalla base di conoscenza non appena ha perso tutte le giustificazioni, una formula viene semplicemente marcata come se fosse assente. Se un'asserzione successiva ripristina una delle sue giustificazioni, si può marcare di nuovo la formula come presente. In questo modo il JTMS conserva tutte le catene di inferenza utilizzate e non deve derivare daccapo le formule una volta che sono nuovamente giustificate.

Oltre a gestire la ritrattazione di informazioni scorrette, i TMS possono essere usati per velocizzare l'analisi di diverse situazioni ipotetiche. Supponiamo che il Comitato Olimpico Rumeno stia scegliendo i luoghi per gli eventi di nuoto, atletica ed equitazione ai Giochi Olimpici del 2048, che si terranno in Romania. Supponiamo che la prima ipotesi sia *Sito(Nuoto, Pitesti)*, *Sito(Atletica, Bucarest)* e *Sito(Equitazione, Arad)*. Per calcolare tutte le conseguenze logistiche e quindi l'opportunità di questa scelta occorrerà svolgere una gran mole di ragionamento. Se poi vogliamo considerare la scelta *Sito(Atletica, Sibiu)*, il TMS ci permetterà di non ricalcolare tutto daccapo: basterà ritrattare *Sito(Atletica, Bucarest)* e asserire al suo posto *Sito(Atletica, Sibiu)*; il TMS si prenderà cura di tutte le revisioni necessarie. Le catene di inferenza generate quando la scelta era caduta su Bucarest potranno essere riusate nel caso di Sibiu, a patto che le conclusioni siano le stesse.

Un sistema di mantenimento di verità basato su assunzioni, o **ATMS** (*assumption-based truth maintenance system*), è progettato per rendere particolarmente efficiente questo tipo di "cambio di contesto" tra mondi ipotetici. In un JTMS, la memorizzazione delle giustifi-

**sistema
di mantenimento
della verità**

JTMS
giustificazione

ATMS

spiegazione**assunzione**

cazioni permette di muoversi rapidamente da uno stato a un altro con qualche ritrattazione e asserzione, ma in ogni istante è rappresentato un solo stato. Un ATMS rappresenta contemporaneamente *tutti* gli stati considerati. Laddove un JTMS si limita a etichettare ogni formula come *in* o *out* dalla KB, un ATMS tiene traccia, per ogni formula, di quali assunzioni la renderebbero vera. In altre parole, ogni formula è etichettata da un insieme di insiemi di assunzioni. Una formula è vera solo nei casi in cui tutte le assunzioni di uno degli insiemi di assunzioni sono vere.

I sistemi di mantenimento della verità forniscono anche un meccanismo per generare **spiegazioni**. Tecnicamente, una spiegazione di una formula P è un insieme di formule E tali che da E segue logicamente P . Se le formule in E sono già note come vere, allora E fornisce una base sufficiente per dimostrare che P deve valere. Le spiegazioni tuttavia possono anche includere **assunzioni**: formule che non sono note come vere, ma basterebbero a dimostrare P se lo fossero. Per esempio, se la vostra auto non parte, probabilmente non avete informazioni sufficienti per dimostrare con certezza la causa del problema, ma una spiegazione ragionevole potrebbe includere l'assunzione che la batteria sia scarica. Questo, insieme alla conoscenza del funzionamento delle macchine spiegherebbe il comportamento osservato. Nella maggior parte dei casi preferiremo le spiegazioni minime, intendendo con questo che non deve esistere un sottoinsieme proprio di E che sia anch'esso una spiegazione. Un ATMS può generare spiegazioni per il problema “la macchina non parte” formulando assunzioni (come “serbatoio vuoto” o “batteria scarica”) in qualsiasi ordine desideriamo, anche se alcune di esse sono contraddittorie. Alla fine potremo leggere l'etichetta della formula “la macchina non parte” e guardare gli insiemi di assunzioni che la giustificherebbero.

Gli algoritmi che implementano i sistemi di mantenimento della verità sono un po' complessi, e non li tratteremo qui. La complessità computazionale del problema è almeno pari a quello dell'inferenza proposizionale, ovvero NP-difficile. Di conseguenza, non ci si può aspettare che il mantenimento della verità rappresenti una panacea: usato accortamente, comunque, un TMS può fornire un incremento significativo nella capacità di un sistema logico di gestire ambienti e ipotesi complesse.

10.7 Riepilogo

Avendo esplorato i particolari della rappresentazione di molti tipi di conoscenza, speriamo di aver dato ai lettori un'idea di come sono costruite le basi di conoscenza reali e delle interessanti questioni filosofiche che sorgono. I punti più importanti sono i seguenti.

- La rappresentazione della conoscenza su larga scala richiede un'ontologia generale per organizzare e collegare i diversi domini della conoscenza.
- Un'ontologia generale deve coprire una grande varietà di conoscenze diverse e dovrebbe essere capace, in linea di principio, di gestire qualsiasi dominio.
- Costruire un'ontologia generale e di grandi dimensioni è una sfida importante che non è stata ancora del tutto vinta, anche se le infrastrutture attuali sembrano piuttosto robuste.
- Abbiamo presentato un'**ontologia superiore** basata sulle categorie e sul calcolo degli eventi. Abbiamo trattato le categorie, le sottocategorie, le parti, gli oggetti strutturati, le misure, le sostanze, gli eventi, lo spazio e il tempo, il cambiamento e le credenze.
- I tipi naturali non possono essere definiti in maniera completa nella logica, ma le proprietà dei tipi naturali possono essere rappresentate.
- Le azioni, gli eventi e il tempo possono essere rappresentati con il calcolo degli eventi. Tali rappresentazioni permettono a un agente di costruire sequenze di azioni e di fare inferenze logiche su ciò che sarà vero quando tali azioni saranno eseguite.

- Per organizzare le gerarchie di categorie sono stati sviluppati sistemi di rappresentazione specializzati, come le **reti semantiche** e le **logiche descrittive**. L'**ereditarietà** è una forma importante di inferenza, che permette di dedurre le proprietà degli oggetti dalle categorie a cui appartengono.
- L'**ipotesi del mondo chiuso**, com'è implementata nei programmi logici, rappresenta un modo semplice di evitare di specificare una gran quantità di informazione negativa. Il modo migliore di interpretarla è come un **default** che può essere sovrascritto da informazione aggiuntiva.
- Le **logiche non monotone**, come la **circoscrizione** e la **logica di default**, hanno lo scopo di catturare il ragionamento di default in generale.
- I **sistemi di mantenimento della verità** gestiscono efficientemente gli aggiornamenti e la revisione della conoscenza.
- È difficile costruire manualmente ontologie di grandi dimensioni; si può facilitare il lavoro estraendo conoscenza dai testi.

Note storiche e bibliografiche

Briggs (1985) afferma che la ricerca sulla rappresentazione della conoscenza ebbe inizio in India nel primo millennio a.C. con l'analisi della grammatica del sanscrito shastrico. I filosofi occidentali fanno risalire il loro lavoro su questo soggetto al 300 a.C. con la *Metafisica* di Aristotele (letteralmente, gli scritti successivi al libro sulla fisica). In effetti, lo sviluppo di terminologia tecnica in qualsiasi campo può essere considerata una forma di rappresentazione della conoscenza.

Le prime discussioni sulla rappresentazione nella comunità dell'IA tendevano a focalizzarsi sulla “rappresentazione del *problema*” piuttosto che sulla “rappresentazione della *conoscenza*” (cfr. per esempio la trattazione di Amarel (1968) del problema dei missionari e dei cannibali). Negli anni 1970 ebbe molta importanza lo sviluppo di “sistemi esperti” (chiamati anche “sistemi basati sulla conoscenza”) che potevano, una volta fornita loro appropriata conoscenza del dominio, uguagliare o superare le prestazioni degli esperti umani su compiti molto precisi. Il primo sistema esperto, DENDRAL (Feigenbaum *et al.*, 1971; Lindsay *et al.*, 1980), era in grado di interpretare l'output di uno spettrometro di massa (uno strumento usato per analizzare la struttura dei composti chimici organici) con la stessa accuratezza di chimici esperti. Benché il suo successo sia stato fondamentale per convincere la comunità dell'IA dell'importanza della rappresentazione della conoscenza, i formalismi usati da DENDRAL erano strettamente legati al dominio della chimica.

Col tempo i ricercatori hanno cominciato a rivolgere l'attenzione verso formalismi di rappresentazione

della conoscenza standardizzati e ontologie che potevano facilitare la creazione di nuovi sistemi esperti. Questo li ha portati ad avventurarsi in un territorio precedentemente esplorato dai filosofi della scienza e del linguaggio. La disciplina imposta dall'IA e relativa alla necessità che le teorie “funzionino” ha portato a sviluppi più rapidi e profondi rispetto a quando questi problemi erano esclusivo appannaggio della filosofia (benché talvolta abbia anche portato alla ripetuta reinvenzione della ruota).

Ma fino a che punto possiamo fidarci della conoscenza degli esperti? Già nel 1955 Paul Meehl (cfr. anche Grove e Meehl, 1996) studiò i processi decisionali di esperti istruiti in attività soggettive come la previsione del successo di uno studente in un programma di formazione o la recidività di un criminale. In 19 dei 20 studi di cui fece la revisione, Meehl trovò che semplici algoritmi di apprendimento statistico (come la regressione lineare o bayesiana) ottenevano previsioni migliori degli esperti. Anche Tetlock (2017) studiò la conoscenza degli esperti e concluse che era lacunosa nei casi difficili. L'Educational Testing Service ha usato un programma automatizzato per valutare milioni di risposte testuali aperte a domande del test GMAT (*graduate management admission test*) fin dal 1999. Il programma riporta voti concordi con quelli di esaminatori umani nel 97% dei casi, circa lo stesso livello di concordanza fra due esaminatori umani (Burstein *et al.*, 2001). Questo non significa che il programma comprende le risposte, ma solo che è in grado di distinguere quelle corrette da quelle errate più o meno come gli esaminatori umani.

La creazione di tassonomie complete e di classificazioni risale ai tempi antichi. Aristotele (384–322 a. C.) enfatizzò fortemente gli schemi di classificazione e categorizzazione. Il suo *Organon*, una serie di opere dedicate alla logica raccolte dai suoi discepoli dopo la sua morte, includeva un trattato chiamato *Categorie* in cui cercava di costruire quella che oggi chiameremmo un’ontologia superiore. Aristotele introdusse anche le nozioni di **genere** e **specie** per le classificazioni di livello inferiore. Il nostro sistema di classificazione biologica, che include l’uso della “nomenclatura binomiale” (classificazione per genere e specie in senso tecnico), fu inventato dal biologo svedese Carlo Linneo, o Carl von Linne (1707–1778). I problemi associati ai tipi naturali e ai limiti sfumati tra le categorie sono stati considerati, tra gli altri, da Wittgenstein (1953), Quine (1953), Lakoff (1987) e Schwartz (1977).

Rimandiamo al Capitolo 24 del Volume 2 per una discussione delle rappresentazioni in reti neurali deep di parole e concetti che evitano alcuni dei problemi di una ontologia rigida, ma sacrificando in parte la precisione. Non conosciamo ancora il modo migliore per unire i vantaggi delle reti neurali e della semantica della logica per la rappresentazione.

L’interesse per le ontologie di dimensioni sempre più grandi sta aumentando, come è documentato dall’*Handbook on Ontologies* (Staab, 2004). Il progetto OPENCYC (Lenat e Guha, 1990; Matuszek *et al.*, 2006) ha rilasciato un’ontologia di 150.000 concetti, con un’ontologia superiore simile a quella della Figura 10.1 e concetti specifici quali “OLED Display” e “iPhone”, che è un tipo di “cellular phone” che a sua volta è un tipo di “consumer electronics”, “phone”, “wireless communication device” e altri concetti. Il progetto NEXTKB estende CYC e altre risorse tra cui FrameNet e WordNet in una base di conoscenza con quasi 3 milioni di fatti e fornisce un motore di ragionamento, FIRE, per gestirla (Forbus *et al.*, 2010).

Il progetto DBPEDIA estrae dati strutturati da Wikipedia, e nello specifico dalle Infobox, le coppie attributo/valore che accompagnano molti articoli di Wikipedia (Wu e Weld, 2008; Bizer *et al.*, 2007). Nel 2015, DBPEDIA conteneva 400 milioni di fatti su 4 milioni di oggetti nella sola versione inglese; contando tutte le 110 lingue si arriva a 1,5 miliardi di fatti (Lehmann *et al.*, 2015).

Il gruppo di lavoro IEEE P1600.1 ha creato SUMO, la Suggested Upper Merged Ontology (Niles e Pease, 2001; Pease e Niles, 2002), con circa 1000 termini nell’ontologia superiore e collegamenti a oltre 20.000 termini specifici di vari dominî. Stoffel *et al.*

(1997) descrivono algoritmi per gestire in modo efficiente un’ontologia molto grande. Una rassegna delle tecniche per estrarre conoscenza dalle pagine web è fornita da Etzioni *et al.* (2008).

Sul web stanno emergendo linguaggi di rappresentazione. RDF (Brickley e Guha, 2004) consente di formulare asserzioni sotto forma di triplette relazionali e fornisce alcuni mezzi per evolvere il significato dei nomi nel tempo. OWL (Smith *et al.*, 2004) è una logica descrittiva che supporta inferenze su queste triplette. Finora, l’uso sembra inversamente proporzionale alla complessità rappresentazionale: i tradizionali formati HTML e CSS rappresentano oltre il 99% dei contenuti web, seguiti dai più semplici schemi di rappresentazione, quali RDFa (Adida e Birbeck, 2008) e microformati (Khare, 2006; Patel-Schneider, 2014), che utilizzano tag HTML e XHTML per aggiungere attributi al testo di pagine web. L’uso delle sofisticate ontologie RDF e OWL non è ancora largamente diffuso, e la piena visione del Semantic Web (Berners-Lee *et al.*, 2001) non è stata realizzata. Gli atti dei congressi su *Formal Ontology in Information Systems* (FOIS) trattano ontologie sia generali che specifiche di dominio.

La tassonomia usata in questo capitolo è stata sviluppata dagli autori ed è basata in parte sulla loro esperienza sul progetto CYC e in parte sul lavoro di Hwang e Schubert (1993) e Davis (1990, 2005). Una discussione molto interessante del progetto generale di rappresentazione della conoscenza di senso comune compare nel “Naive Physics Manifesto” di Hayes (1978, 1985b).

Tra le ontologie profonde di successo all’interno di un campo specifico vi sono il progetto Gene Ontology (Gene Ontology Consortium, 2008) e Chemical Markup Language (Murray-Rust *et al.*, 2003). Dubbi sulla fattibilità di una singola ontologia per *tutta* la conoscenza sono stati espressi da Doctorow (2001), Gruber (2004), Halevy *et al.* (2009) e Smith (2004).

Il calcolo degli eventi è stato introdotto da Kowalski e Sergot (1986) per gestire il tempo continuo, e ne sono state proposte molte variazioni (Sadri e Kowalski, 1995; Shanahan, 1997) e panoramiche (Shanahan, 1999; Mueller, 2006). James Allen ha introdotto gli intervalli di tempo (Allen, 1984), sostenendo che fossero molto più naturali delle situazioni per ragionare su eventi prolungati e concorrenti. In Van Lambalgen e Hamm (2005) vediamo come la logica degli eventi corrisponda al linguaggio che utilizziamo per parlare di eventi. Un’alternativa al calcolo degli eventi e delle situazioni è il calcolo dei fluenti (Thielscher, 1999), che reifica i fatti in base agli stati di cui sono composti.

Peter Ladkin (1986a, 1986b) ha proposto la nozione di intervalli temporali “concavi” (cioè con interruzioni; sostanzialmente sono equivalenti a un’unione di normali intervalli “convessi”) e ha applicato alla rappresentazione del tempo le tecniche dell’algebra astratta. Allen (1991) investiga in modo sistematico la grande varietà di tecniche disponibili per la rappresentazione del tempo; van Beek e Manchak (1996) analizzano algoritmi per il ragionamento temporale. Ci sono molti punti in comune tra l’ontologia basata su eventi presentata in questo capitolo e l’analisi degli eventi del filosofo Donald Davidson (1980). Anche le **storie** nell’ontologia dei liquidi di Pat Hayes (1985a) e le **cronache** nella teoria dei piani di McDermott (1985) hanno avuto un importante impatto sulla disciplina e su questo capitolo.

La questione dell’ontologia delle sostanze ha una lunga storia. Platone ha proposto di considerarle entità astratte completamente distinte dagli oggetti fisici; avrebbe detto *FattoDi(Burro₃, Burro)* piuttosto che *Burro₃ ∈ Burro*. Questo conduce a una gerarchia di sostanze in cui, per esempio, *BurroSalato* è una sostanza più specifica di *Burro*. La posizione che abbiamo adottato nel capitolo, e cioè che le sostanze sono categorie di oggetti, fu difesa strenuamente da Richard Montague (1973) ed è stata adottata dal progetto CYC. Copeland (1993) la sottopone a una critica molto severa, ma non priva di punti deboli. L’approccio alternativo che abbiamo menzionato, in cui il burro è un solo oggetto che consiste in tutti gli oggetti burrosi dell’universo, è stata proposta originariamente dal logico polacco Leśniewski (1916). La sua **mereologia** (il nome deriva dal termine greco per “parte”) usava la relazione parte-tutto in sostituzione della teoria matematica degli insiemi, allo scopo di eliminare del tutto entità astratte come gli insiemi stessi. Leonard e Goodman (1940) forniscono un’esposizione di più facile lettura di queste idee, mentre *The Structure of Appearance* (1977), dello stesso Goodman, le applica a vari problemi di rappresentazione della conoscenza. Benché alcuni suoi aspetti siano alquanto scosognati – per esempio, la necessità di ricorrere a un meccanismo di ereditarietà separato basato sulle relazioni parti-tutto – l’approccio mereologico fu capace di guadagnarsi il supporto di Quine (1960). Harry Bunt (1985) ha fornito un’analisi approfondita del suo utilizzo nella rappresentazione della conoscenza. Casati e Varzi (1999) trattano parti, interi e una teoria generale delle posizioni nello spazio.

Lo studio degli oggetti mentali è affrontato con tre approcci principali. Quello adottato in questo capitulo,

lo, basato sulla logica modale e i mondi possibili, è l’approccio classico della filosofia (Hintikka, 1962; Kripke, 1963; Hughes e Cresswell, 1996). Il libro *Reasoning about Knowledge* (Fagin et al., 1995) fornisce un’introduzione completa, mentre Gordon e Hobbs (2017) offrono una teoria formale della psicologia del senso comune.

Il secondo approccio è una teoria del primo ordine in cui gli oggetti mentali sono fluenti, ed è descritto da Davis (2005) e Davis e Morgenstern (2005); si basa sul formalismo dei mondi possibili e sul lavoro di Robert Moore (1980, 1985).

Il terzo approccio è una **teoria sintattica**, in cui gli oggetti mentali sono rappresentati da stringhe di caratteri. Una stringa è semplicemente un termine complesso che denota una lista di simboli, perciò *PuòVolare(Clark)* può essere rappresentato dalla lista di simboli [*P, u, ò, V, o, l, a, r, e, (, C, l, a, r, k,)*]. La teoria sintattica degli oggetti mentali è stata studiata inizialmente da Kaplan e Montague (1960), che hanno mostrato che può portare a dei paradossi se non viene gestita con attenzione. Ernie Davis (1990) offre un eccellente confronto delle teorie della conoscenza sintattica e modale. Pnueli (1977) descrive una logica temporale usata per ragionare sui programmi, in un lavoro che gli ha consentito di vincere il Turing Award e che è stato poi esteso da Vardi (1996). Littman et al. (2017) mostrano che una logica temporale può essere un linguaggio adatto per specificare gli obiettivi a un robot che usa apprendimento con rinforzo in modo che risulti facile agli esseri umani, e che si generalizza bene a diversi ambienti.

Il filosofo greco Porfirio (c. 234–305 a. C.), commentando le *Categorie* di Aristotele, disegnò quella che si potrebbe definire la prima rete semantica. Charles S. Peirce (1909) sviluppò i grafi esistenziali come primo formalismo di rete semantica basato sulla logica moderna. Ross Quillian (1961), spinto dall’interesse verso la memoria umana e l’elaborazione del linguaggio, diede inizio alla ricerca sulle reti semantiche all’interno dell’IA. Un importante articolo di Marvin Minsky (1975) presentò una versione di rete semantica chiamata **frame**; un frame è la rappresentazione di un oggetto o categoria, con attributi e relazioni con altri oggetti e categorie. La questione della semantica sorse in modo prepotente in relazione alle reti semantiche di Quillian (e di quelli che avevano seguito il suo approccio) data la presenza ubiqua di “collegamenti IS-A” (cioè “collegamenti è-un”) estremamente vaghi. Il famoso articolo di Bill Woods (1975) “What’s In a Link?” portò all’attenzione dei ricercatori la necessità

di definire una semantica precisa per i formalismi di rappresentazione della conoscenza. Ron Brachman (1979) discusse questo aspetto e propose alcune soluzioni. L'articolo di Patrick Hayes (1979) "The Logic of Frames" andò ancora più a fondo, sostenendo che "gran parte dei 'frame' non è altro che nuova sintassi per la logica del primo ordine". Drew McDermott's (1978b) in "Tarskian Semantics, or, No Notation without Denotation!" argomentò che l'approccio alla semantica usato nella logica del primo ordine, basato sulla teoria dei modelli, dovrebbe essere applicato a tutti i formalismi per la rappresentazione della conoscenza. Quest'idea rimane controversa; è da notare che lo stesso McDermott ha poi ritrattato la sua posizione in "A Critique of Pure Reason" (McDermott, 1987). Selman e Levesque (1993) discutono la complessità dell'ereditarietà in presenza di eccezioni, mostrando che nella maggior parte delle formulazioni è NP-completa.

Le logiche descrittive sono state sviluppate come sottoinsiemi utili della logica del primo ordine per cui l'inferenza fosse computazionalmente trattabile. Hector Levesque e Ron Brachman (1987) hanno mostrato che alcuni usi della disgiunzione e della negazione erano i principali responsabili dell'intrattabilità dell'inferenza logica. Questo ha portato a una migliore comprensione della relazione fra complessità ed espressività nei sistemi di ragionamento. Calvanese *et al.* (1999) presentano un riassunto dello stato dell'arte, e Baader *et al.* (2007) presentano un manuale completo sulla logica descrittiva.

I tre formalismi principali per la gestione dell'inferenza non monotona – circoscrizione (McCarthy, 1980), logica di default (Reiter, 1980) e logica modale non monotona (McDermott e Doyle, 1980) – furono tutti presentati in un numero speciale dell'*AI Journal*. Delgrande e Schaub (2003) discutono i pregi delle varianti, con 25 anni di senno del poi. L'*answer set programming* può essere considerato un'estensione della negazione come fallimento o come un raffinamento della circoscrizione; la sottostante teoria della semantica dei modelli stabili fu introdotta da Gelfond e Lifschitz (1988); i sistemi più importanti sono DLV (Eiter *et al.*, 1998) e SMODELS (Niemelä *et al.*, 2000). Lifschitz (2001) discute l'applicazione alla pianificazione dell'*answer set programming*. Brewka *et al.* (1997) forniscono una buona panoramica dei diversi approcci alla logica non monotona. Una varietà di sistemi di ragionamento non monotoni basati sulla programmazione logica sono documentati negli atti dei congressi su *Logic Programming and Nonmonotonic Reasoning* (LPNMR).

Lo studio dei sistemi di mantenimento della verità ebbe inizio con TMS (Doyle, 1979) e RUP (McAllester, 1980), che erano entrambi essenzialmente dei JTMS. Forbus e de Kleer (1993) spiegano nei particolari come si possono usare i TMS nelle applicazioni di IA. Nayak e Williams (1997) mostrano come un TMS efficiente rende praticabile la pianificazione in tempo reale delle operazioni di un veicolo spaziale della NASA.

Questo capitolo non copre *ogni* area della rappresentazione della conoscenza. I tre argomenti principali che abbiamo omesso sono i seguenti.

- **Fisica qualitativa:** la fisica qualitativa è una branca della rappresentazione della conoscenza che si occupa specificatamente della costruzione di una teoria logica e non numerica degli oggetti e dei processi fisici. Il termine è stato coniato da Johan de Kleer (1975), benché si possa dire che la ricerca abbia avuto inizio con il sistema BUILD di Fahlman (1974), un sofisticato pianificatore per la costruzione di complesse torri fatte di blocchi. Durante la sua progettazione Fahlman scoprì che la maggior parte dello sforzo (nella sua stima l'80%) era rivolta alla modellazione della fisica del mondo per calcolare la stabilità di vari sotto-aggregati di blocchi piuttosto che alla pianificazione in sé. Di conseguenza delineò un ipotetico processo di "fisica intuitiva" (*naive physics*) per spiegare perché i bambini piccoli possono risolvere problemi come quelli di BUILD senza avere accesso alla sua veloce aritmetica in virgola mobile, necessaria per la modellazione fisica in BUILD. Hayes (1985a) usa le "storie" – sezioni quadridimensionali di spazio-tempo analoghe agli eventi di Davidson – per costruire una fisica intuitiva dei liquidi abbastanza complessa. Davis (2008) fornisce un aggiornamento dell'ontologia dei liquidi che descrive il versamento di liquidi in contenitori.

De Kleer e Brown (1985), Ken Forbus (1985) e Benjamin Kuipers (1985) lavorando in modo indipendente svilupparono, quasi nello stesso tempo, sistemi in grado di ragionare su un sistema fisico basato su astrazioni qualitative delle equazioni sottostanti. La fisica qualitativa si è sviluppata al punto che è possibile analizzare una varietà impressionante di sistemi fisici complessi (Yip, 1991). Le tecniche qualitative sono state applicate alla costruzione di progetti innovativi per orologi, tergilicristalli e robot a sei zampe (Subramanian e Wang, 1994). La raccolta *Readings in Qualitative Reasoning about*

Physical Systems (Weld e de Kleer, 1990), un lemma enciclopedico di Kuipers (2001) e un contributo di Davis riportato in un manuale (2007) forniscono una buona introduzione.

- **Ragionamento spaziale:** il ragionamento necessario per navigare nel mondo del wumpus è banale in confronto alla ricca struttura spaziale del mondo reale. I primi tentativi seri di catturare un ragionamento “di senso comune” riguardante lo spazio si possono trovare nel lavoro di Ernest Davis (1986, 1990). Il calcolo dei collegamenti tra regioni di Cohn *et al.* (1997) supporta una forma di ragionamento spaziale qualitativo e ha condotto a nuovi tipi di sistemi informativi territoriali; cfr. anche (Davis, 2006). Come nel caso della fisica qualitativa, un agente può andare molto lontano, per così dire, senza ricorrere a una rappresentazione metrica completa.
- **Ragionamento psicologico:** il ragionamento psicologico richiede lo sviluppo di una *psicologia* funzionante per agenti artificiali, utilizzabile per ragionare su se stessi e sugli altri agenti. Spesso si basa sulla cosiddetta “psicologia popolare” (*folk psychology*), quella che si ritiene che le persone usino per ragionare su se stesse e sugli altri esseri umani. Le teorie psicologiche che i ricercatori forniscono ai loro agenti artificiali per ragionare sugli altri agenti sono spesso basate sulla descrizione del pro-

getto degli stessi agenti logici. Oggi come oggi il ragionamento psicologico è utilizzato soprattutto nel contesto della comprensione del linguaggio naturale, in cui la valutazione delle intenzioni del parlante ha un’importanza fondamentale.

Minker (2001) raccoglie articoli pubblicati dai più importanti studiosi della rappresentazione della conoscenza, riassumendo 40 anni di lavoro. Le fonti più aggiornate sono costituite dagli atti dei congressi internazionali *Principles of Knowledge Representation and Reasoning. Readings in Knowledge Representation* (Brachman e Levesque, 1985) e *Formal Theories of the Commonsense World* (Hobbs e Moore, 1985) sono due eccellenti antologie; la prima si concentra principalmente sugli articoli di importanza storica sui linguaggi e i formalismi di rappresentazione, la seconda sul processo di accumulo della conoscenza stessa. Davis (1990), Stefik (1995) e Sowa (1999) sono libri di testo introduttivi alla rappresentazione della conoscenza, van Harmelen *et al.* (2007) forniscono un manuale, e Davis e Morgenstern (2004) hanno curato un numero speciale di *AI Journal* sull’argomento. Davis (2017) fornisce una panoramica sulla logica per il ragionamento di senso comune. La conferenza biennale *Theoretical Aspects of Reasoning About Knowledge* (TARK) tratta applicazioni della teoria della conoscenza in IA, economia e sistemi distribuiti.

Pianificazione automatica

- 11.1 Definizione di pianificazione classica
- 11.2 Algoritmi di pianificazione classica
- 11.3 Euristiche per la pianificazione
- 11.4 Pianificazione gerarchica
- 11.5 Pianificazione e azione in ambienti non deterministici
- 11.6 Tempo, scheduling e risorse
- 11.7 Analisi degli approcci alla pianificazione
- 11.8 Riepilogo
Note storiche e bibliografiche

In cui vediamo come un agente può sfruttare la struttura di un problema per costruire in modo efficiente complessi piani d'azione.

Pianificare un corso d'azione è un requisito fondamentale per un agente intelligente. Scegliere la rappresentazione corretta per azioni e stati e gli algoritmi giusti può facilitare il compito. Nel Paragrafo 11.1 introduciamo un linguaggio generale per la rappresentazione **fattorizzata** di problemi di pianificazione che è in grado di rappresentare in modo naturale e sintetico un'ampia varietà di domini, può essere adattato in modo efficiente anche a problemi di grandi dimensioni e non richiede euristiche ad hoc per un nuovo dominio. Il Paragrafo 11.4 estende il linguaggio di rappresentazione per consentire azioni gerarchiche, così da permetterci di affrontare problemi più complessi. Nel Paragrafo 11.2 esaminiamo gli algoritmi efficienti per la pianificazione e nel Paragrafo 11.3 trattiamo le heuristiche per tali algoritmi. Nel Paragrafo 11.5 affrontiamo i domini parzialmente osservabili e non deterministici, e nel Paragrafo 11.6 estendiamo ulteriormente il linguaggio per gestire problemi di scheduling con vincoli di risorse, avvicinandoci così ai sistemi di pianificazione utilizzati nel mondo reale per pianificare e programmare le attività di veicoli spaziali, fabbriche e campagne militari. Il Paragrafo 11.7 analizza l'efficacia di queste tecniche.

11.1 Definizione di pianificazione classica

pianificazione classica

La **pianificazione classica** consiste per definizione nel trovare una sequenza di azioni per raggiungere un obiettivo in un ambiente discreto, deterministico, statico, completamente osservabile. Abbiamo visto due approcci a questa attività: l’agente per la risoluzione di problemi del Capitolo 3 e l’agente logico proposizionale ibrido del Capitolo 7. Entrambi questi approcci condividono due limitazioni: in primo luogo, richiedono entrambi euristiche ad hoc per ogni nuovo dominio: una funzione di valutazione euristica per la ricerca e codice scritto manualmente per l’agente del wumpus ibrido; in secondo luogo, entrambi gli approcci necessitano di rappresentare in modo esplicito uno spazio degli stati esponenzialmente grande. Per esempio, nel modello logico proposizionale del mondo del wumpus l’assioma per muovere un passo avanti deve essere ripetuto per tutti e quattro gli orientamenti dell’agente, T passi temporali e n^2 posizioni correnti.

PDDL

In risposta a queste limitazioni, gli studiosi di pianificazione hanno definito una **rappresentazione fattorizzata** utilizzando un linguaggio denominato **PDDL**, *planning domain definition language* (Ghallab *et al.*, 1998), che consente di esprimere tutte le $4Tn^2$ azioni con un unico schema di azione e non richiede una conoscenza specifica del dominio. Esiste una versione di base di PDLL che consente di gestire domini di pianificazione classica ed estensioni in grado di gestire domini non classici che sono continui, parzialmente osservabili, correnti e multiagente. La sintassi del linguaggio PDLL è basata sul Lisp, ma la tradurremo in una forma corrispondente alla notazione usata in questo libro.

stato

In PDLL uno **stato** è rappresentato come una congiunzione di fluenti che sono atomi ground. Ricordiamo che “ground” significa senza variabili e “fluente” indica un aspetto del mondo che cambia nel tempo, mentre “atomo ground” significa che esiste un singolo predicato e che, se ci sono argomenti, devono essere costanti. Per esempio, *Povero* \wedge *Sconosciuto* potrebbe rappresentare lo stato di un agente sfortunato, mentre uno stato di un problema di consegna di pacchi potrebbe essere *Posizione(Camion₁, Melbourne)* \wedge *Posizione(Camion₂, Sydney)*. PDLL usa la **semantica dei database**: l’ipotesi del mondo chiuso significa che qualsiasi fluente non menzionato è falso, e l’ipotesi dei nomi unici significa che *Camion₁* e *Camion₂* sono distinti.

I fluenti seguenti *non* sono consentiti in uno stato: *Posizione(x, y)* (perché è non ground, dato che ha delle variabili), \neg *Povero* (perché è una negazione) e *Posizione(Moglie(Ali), Sydney)* (perché utilizza un simbolo di funzione, *Moglie*). Quando è utile possiamo pensare alla congiunzione di fluenti come a un *insieme* di fluenti.

schema di azione

Uno **schema di azione** rappresenta una famiglia di azioni ground. Per esempio, ecco uno schema di azione per far volare un aereo da un luogo a un altro:

```
Azione(Vola(p, da, a),
       PRECOND:Posizione(p, da)  $\wedge$  Aereo(p)  $\wedge$  Aeroporto(da)  $\wedge$  Aeroporto(a)
       EFFETTO: $\neg$ Posizione(p, da)  $\wedge$  Posizione(p, a)).
```

precondizione effetto

Lo schema è costituito da un nome di azione, un elenco di tutte le variabili utilizzate, una **precondizione** e un **effetto**. La precondizione e l’effetto sono entrambi congiunzioni di letterali (formule atomiche positive o negative). Possiamo scegliere delle costanti per istanziare le variabili, ottenendo così un’azione ground (senza variabili):

```
Azione(Vola(P1, SFO, JFK),
       PRECOND:Posizione(P1, SFO)  $\wedge$  Aereo(P1)  $\wedge$  Aeroporto(SFO)  $\wedge$  Aeroporto(JFK)
       EFFETTO: $\neg$ Posizione(P1, SFO)  $\wedge$  Posizione(P1, JFK)).
```

Un’azione ground *a* è **applicabile** nello stato *s* se da *s* consegue logicamente la precondizione di *a*, cioè se ogni letterale positivo nella precondizione è in *s* e ogni letterale negato non è in *s*.

Il **risultato** dell'esecuzione dell'azione a nello stato s è definito come lo stato s' rappresentato dall'insieme di fluenti formato partendo da s , rimuovendo i fluenti che appaiono come letterali negativi negli effetti dell'azione (che chiamiamo **lista di eliminazioni** o $\text{DEL}(a)$) e aggiungendo i fluenti che sono letterali positivi negli effetti dell'azione (che chiamiamo **lista di aggiunte** o $\text{ADD}(a)$):

$$\text{RISULTATO}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a). \quad (11.1)$$

Per esempio, con l'azione $Vola(P_1, SFO, JFK)$ elimineremmo il fluente $\text{Posizione}(P_1, SFO)$ e aggiungeremmo il fluente $\text{Posizione}(P_1, JFK)$.

Un insieme di schemi di azione serve quale definizione di un *dominio* di pianificazione. Un problema *specifico* nel dominio è definito con l'aggiunta di uno stato iniziale e di un obiettivo. Lo **stato iniziale** è una congiunzione di fluenti ground (nella Figura 11.1 è introdotto con la parola chiave *Init*). Come per tutti gli stati, si utilizza l'ipotesi del mondo chiuso, che significa che ogni atomo non menzionato è falso. L'**obiettivo** (introdotto con *Obiettivo* nella Figura 11.1) è come una precondizione: una congiunzione di letterali (positivi o negativi) che può contenere variabili. Per esempio, l'obiettivo $\text{Posizione}(C_1, SFO) \wedge \neg\text{Posizione}(C_2, SFO) \wedge \neg\text{Posizione}(p, SFO)$ indica qualsiasi stato in cui il carico merci C_1 si trova in SFO mentre C_2 no e in cui c'è un aereo in SFO .

lista di eliminazioni
lista di aggiunte

11.1.1 Esempio: trasporto aereo di merci

La Figura 11.1 presenta un problema di trasporto aereo che coinvolge il carico/scarico di merci su aerei e il volo di questi ultimi da un posto all'altro. Il problema può essere definito con tre azioni: *Carica*, *Scarica* e *Vola*. Le azioni hanno effetto su due predicati: $\text{In}(c, p)$ significa che le merci c si trovano sull'aereo p ; $\text{Posizione}(x, a)$ significa che l'oggetto x (un aereo o delle merci) sono all'aeroporto a . Notate che occorre prestare attenzione ad assicurarsi che i predicati *Posizione* siano gestiti in maniera appropriata. Quando un aereo vola da un aeroporto a un altro, tutte le merci al suo interno viaggiano con esso. Nella logica del primo ordine sarebbe facile quantificare su tutti gli oggetti all'interno dell'aereo, ma il linguaggio PDDL non ha un quantificatore universale, perciò ci serve una soluzione diversa. Il nostro approccio consiste nel dire che una merce cessa di essere in una *Posizione* quando è *In* un aereo; la merce si trova nella *Posizione* del nuovo aeroporto soltanto quando viene scaricata.

```

Init(Posizione(C1, SFO) ∧ Posizione(C2, JFK) ∧ Posizione(P1, SFO) ∧ Posizione(P2, JFK)
    ∧ Merci(C1) ∧ Merci(C2) ∧ Aereo(P1) ∧ Aereo(P2)
    ∧ Aeroporto(JFK), Aeroporto(SFO))
Obiettivo(Posizione(C1, JFK) ∧ Posizione(C2, SFO))
Azione(Carica(c, p, a),
    PRECOND: Posizione(c, a) ∧ Posizione(p, a) ∧ Merci(c) ∧ Aereo(p) ∧ Aeroporto(a)
    EFFETTO: ¬Posizione(c, a) ∧ In(c, p))
Azione(Scarica(c, p, a),
    PRECOND: In(c, p) ∧ Posizione(p, a) ∧ Merci(c) ∧ Aereo(p) ∧ Aeroporto(a)
    EFFETTO: Posizione(c, a) ∧ ¬In(c, p))
Azione(Vola(p, da, a),
    PRECOND: Posizione(p, da) ∧ Aereo(p) ∧ Aeroporto(da) ∧ Aeroporto(a)
    EFFETTO: ¬Posizione(p, da) ∧ Posizione(p, a))

```

Figura 11.1 Una descrizione PDDL di un problema di trasporto aereo di merci.

```

Init(Gomma(Bucata) ∧ Gomma(Scorta) ∧ Posizione(Bucata, Asse) ∧ Posizione(Scorta, Bagagliaio))
Obiettivo(Posizione(Scorta, Asse))
Azione(Rimuovi(ogg, pos)
    PRECOND: Posizione(ogg, pos)
    EFFETTO : ¬Posizione(ogg, pos) ∧ Posizione(ogg, Terreno))
Azione(Monta(t, Asse)
    PRECOND: Gomma(t) ∧ Posizione(t, Terreno) ∧ ¬Posizione(Bucata, Asse) ∧ ¬Posizione(Scorta, Asse)
    EFFETTO: ¬Posizione(t, Terreno) ∧ Posizione(t, Asse))
Azione(AbbandonaDiNotte,
    PRECOND:
    EFFETTO: ¬Posizione(Scorta, Terreno) ∧ ¬Posizione(Scorta, Asse) ∧ ¬Posizione(Scorta, Bagagliaio)
        ∧ ¬Posizione(Bucata, Terreno) ∧ ¬Posizione(Bucata, Asse) ∧ ¬Posizione(Bucata, Bagagliaio))

```

Figura 11.2 Il semplice problema della ruota bucata.

Perciò *Posizione* in effetti significa “disponibile per l’uso in una data posizione”. Il piano seguente è una soluzione del problema:

[Carica(C_1, P_1, SFO), Vola(P_1, SFO, JFK), Scarica(C_1, P_1, JFK),
Carica(C_2, P_2, JFK), Vola(P_2, JFK, SFO), Scarica(C_2, P_2, SFO)].

11.1.2 Esempio: il problema della ruota di scorta

Considerate il problema di cambiare una ruota bucata (Figura 11.2). L’obiettivo consiste nell’avere una ruota di scorta montata in maniera corretta sull’asse della macchina, mentre lo stato iniziale ha una ruota bucata sull’asse e una ruota di scorta nel bagagliaio. Per semplicità la nostra versione del problema è molto astratta e non prevede bulloni incastrati o altre complicazioni. Le azioni sono solo quattro: prendere la ruota di scorta dal bagagliaio, rimuovere quella bucata dall’asse, montare la ruota di scorta sull’asse e lasciare la macchina incustodita tutta la notte. Presumiamo che la macchina sia parcheggiata in un quartiere particolarmente malfamato, per cui l’effetto di lasciarla incustodita è la sparizione di tutte le ruote. Una soluzione del problema è [*Rimuovi(Bucata, Asse)*, *Rimuovi(Scorta, Bagagliaio)*, *Monta(Scorta, Asse)*].

11.1.3 Esempio: il mondo dei blocchi

Uno dei più famosi domini di pianificazione è il **mondo dei blocchi**. Questo dominio consiste in un insieme di blocchi di forma cubica appoggiati a un tavolo arbitrariamente grande.¹ I blocchi possono essere impilati ma solo un blocco può stare direttamente sopra un altro. Un braccio robotico può prendere un blocco e spostarlo in un’altra posizione o sul tavolo oppure su un altro blocco. Il braccio può tenere un solo blocco alla volta, per cui non può afferrarne uno che ne ha sopra un altro. Un obiettivo tipico può essere quello di avere il blocco *A* su quello *B* e il blocco *B* su quello *C* (Figura 11.3).

Useremo *Sopra(b, x)* per indicare che il blocco *b* si trova su *x*, dove *x* può essere un altro blocco o il tavolo. L’azione di muovere il blocco *b* da sopra *x* a sopra *y* si scriverà *Muovi(b, x, y)*. Ora, una delle precondizioni per muovere *b* è che non ci sia un blocco su di

¹ Il mondo dei blocchi utilizzato nella ricerca sulla pianificazione è molto più semplice della versione di SHRDLU mostrata nella Figura 1.3 (Capitolo 1).

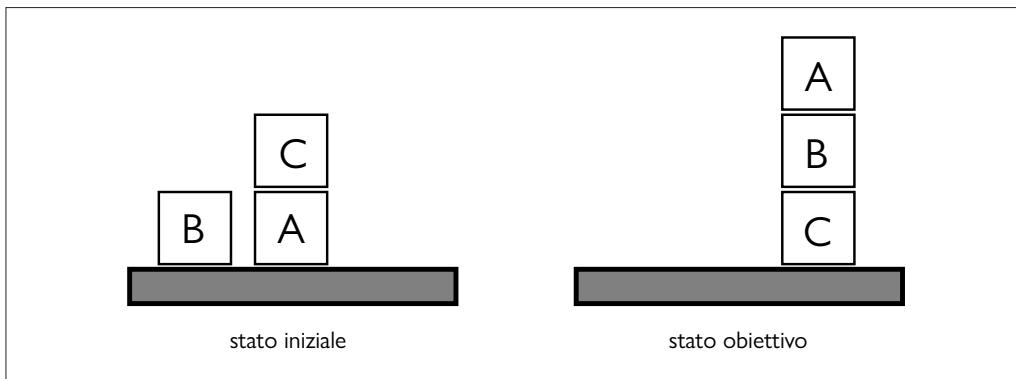


Figura 11.3
Rappresentazione del problema del mondo dei blocchi specificato nella Figura 11.4.

Init(Sopra(A, Tavolo) \wedge Sopra(B, Tavolo) \wedge Sopra(C, A))

\wedge Blocco(A) \wedge Blocco(B) \wedge Blocco(C)

\wedge Libero(B) \wedge Libero(C) \wedge Libero(Tavolo))

Obiettivo(Sopra(A, B) \wedge Sopra(B, C))

Azione(Muovi(b, x, y),

PRECOND: *Sopra(b, x) \wedge Libero(b) \wedge Libero(y) \wedge Blocco(b) \wedge Blocco(y) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)*

EFFETTO: *Sopra(b, y) \wedge Libero(x) \wedge \neg Sopra(b, x) \wedge \neg Libero(y))*

Azione(MuoviSulTavolo(b, x),

PRECOND: *Sopra(b, x) \wedge Libero(b) \wedge Blocco(b) \wedge Blocco(x) ,*

EFFETTO: *Sopra(b, Tavolo) \wedge Libero(x) \wedge \neg Sopra(b, x))*

Figura 11.4 Un problema di pianificazione nel mondo dei blocchi: la costruzione di una torre di tre blocchi. Una soluzione è costituita dalla sequenza [MuoviSulTavolo(C,A)Muovi(B, Tavolo, C), Muovi(A, Tavolo, B)].

esso. Nella logica del primo ordine scriveremmo $\neg \exists x \text{ Sopra}(x, b)$ o, alternativamente, $\forall x \neg \text{Sopra}(x, b)$. Il linguaggio PDDL non consente quantificatori, perciò introduciamo un predicato *Libero(x)*, che è vero quando non c'è nulla sopra *x* (la descrizione completa del problema è riportata nella Figura 11.4).

L'azione *Muovi* sposta il blocco *b* da *x* a *y* se sia *b* che *y* sono liberi. Alla fine della mossa *b* sarà libero, ma *y* no. Un primo tentativo per lo schema *Muovi* è:

Azione(Muovi(b, x, y),

PRECOND: *Sopra(b, x) \wedge Libero(b) \wedge Libero(y),*

EFFETTO: *Sopra(b, y) \wedge Libero(x) \wedge \neg Sopra(b, x) \wedge \neg Libero(y)) .*

Sfortunatamente, questo schema non gestisce propriamente *Libero* quando *x* o *y* è il tavolo. Quando *x* è il *Tavolo*, l'azione ha come effetto *Libero(Tavolo)*, ma il tavolo non dovrebbe diventare libero; quando invece *y = Tavolo* risulta una precondizione *Libero(Tavolo)*, ma il tavolo non dev'essere libero per potervi spostare un blocco. Per correggere quest'errore occorre fare due cose. Per prima cosa introduciamo un'altra azione per spostare un blocco *b* da *x* al tavolo:

Azione(MuoviSulTavolo(b, x),

PRECOND: *Sopra(b, x) \wedge Libero(b)),*

EFFETTO: *Sopra(b, Tavolo) \wedge Libero(x) \wedge \neg Sopra(b, x)) .*

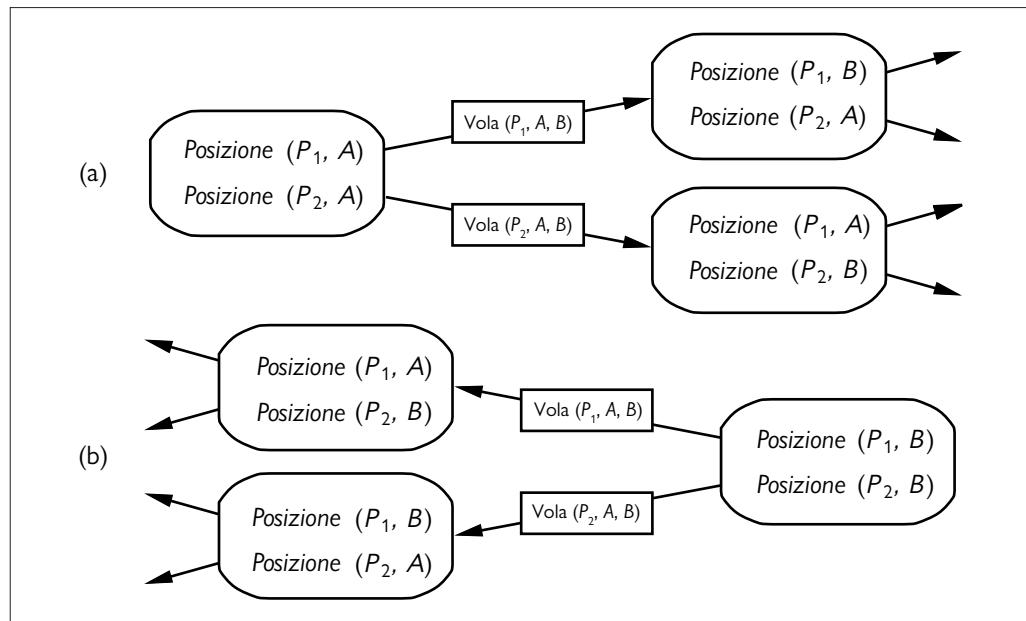


Figura 11.5 Due approcci per la ricerca di un piano. (a) La ricerca in avanti (progressione) nello spazio degli stati ground, che parte dallo stato iniziale e usa le azioni del problema per cercare in avanti un membro dell'insieme degli stati obiettivo. (b) La ricerca all'indietro (regressione) cerca attraverso le descrizioni degli stati, iniziando dall'obiettivo e utilizzando l'inverso delle azioni per risalire verso lo stato iniziale.

In secondo luogo, daremo a *Libero*(x) l'interpretazione “c'è uno spazio libero in cima a x abbastanza ampio da appoggiarvi un blocco”. Sotto quest'interpretazione, *Libero*(*Tavolo*) sarà sempre vero. L'unico problema è che nulla impedisce al pianificatore di usare *Muovi*(b , x , *Tavolo*) al posto di *MuoviSulTavolo*(b , x). Possiamo tollerare questo problema: infatti porterà a uno spazio di ricerca più grande del necessario, ma non sarà la causa di risposte scorrette. Alternativamente, potremmo introdurre il predicato *Blocco* e aggiungere *Blocco*(b) \wedge *Blocco*(y) alla precondizione di *Muovi*, come nella Figura 11.4.

11.2 Algoritmi per la pianificazione classica

La descrizione di un problema di pianificazione indica un modo ovvio per effettuare una ricerca a partire dallo stato iniziale attraverso lo spazio degli stati, puntando a un obiettivo. Uno dei vantaggi della rappresentazione dichiarativa degli schemi di azione è che possiamo anche cercare procedendo all'indietro a partire dall'obiettivo, cercando lo stato iniziale (la Figura 11.5 mette a confronto le ricerche in avanti e all'indietro). Una terza possibilità è quella di tradurre la descrizione del problema in un insieme di formule logiche a poter applicare un algoritmo di inferenza logica per trovare una soluzione.

11.2.1 Ricerca in avanti nello spazio degli stati per problemi di pianificazione

Possiamo risolvere i problemi di pianificazione applicando un algoritmo di ricerca euristica come quelli del Capitolo 3 o del Capitolo 4. Gli stati in questo spazio degli stati sono stati ground, dove ogni fluente è vero o falso. L'obiettivo è uno stato che ha tutti i fluenti positivi dell'obiettivo del problema e nessuno dei fluenti negativi. Le azioni effettuabili in uno stato,

Azioni(s), sono istanze ground degli schemi di azione, cioè azioni in cui le variabili sono state tutte sostituite da valori costanti.

Per determinare le azioni applicabili unifichiamo lo stato corrente rispetto alle precondizioni di ciascuno schema di azione. Per ogni unificazione valida che porta a una sostituzione, applichiamo la sostituzione allo schema di azione per ottenere un’azione ground senza variabili (è un requisito degli schemi di azione che ogni variabile nell’effetto debba apparire anche nella precondizione; in tal modo vi è la garanzia che non rimanga alcuna variabile dopo la sostituzione).

Per ciascuno schema possono esistere più modalità di unificazione. Nell’esempio della gomma bucata (Paragrafo 11.1.2) l’azione *Rimuovi* ha la precondizione *Posizione(ogg, pos)*, che può corrispondere allo stato iniziale in due modi che portano alle due sostituzioni: *{ogg/Bucata, pos/Asse}* e *{ogg/Scorta, pos/Bagagliaio}*; applicando queste sostituzioni ottieniamo due azioni ground. Se un’azione ha più letterali nella precondizione, ognuno di essi può essere fatto corrispondere allo stato corrente in più modi.

A prima vista sembra che lo spazio degli stati possa essere troppo grande per molti problemi. Considerate un problema di trasporto aereo con 10 aeroporti con 5 aerei e 20 pacchi di merci per aeroporto. L’obiettivo è spostare tutte le merci dall’aeroporto *A* a quello *B*. Esiste una soluzione in 41 passi: caricare tutti e 20 i pacchi su uno degli aerei in *A*, volare in *B* e scaricare tutti i 20 pacchi.

Trovare questa soluzione apparentemente facile può invece risultare difficile, perché il fattore di ramificazione è enorme: ognuno dei 50 aerei può volare in 9 altri aeroporti, e ognuno dei 200 pacchi può essere scaricato (se si trova su un aereo) o caricato in uno qualsiasi degli aerei presenti (se si trova in un aeroporto). Perciò in ogni stato ci sono da un minimo di 450 azioni (quando tutti i pacchi sono in aeroporti senza aerei) a un massimo di 10.450 (quando tutti i pacchi e gli aerei si trovano nello stesso aeroporto). In media, diciamo che le azioni possibili sono circa 2000 per stato, perciò la profondità del grafo di ricerca fino alla soluzione in 41 passi avrà circa 2000⁴¹ nodi.

È chiaro che anche questa istanza di problema relativamente piccola non ha speranza di essere risolta senza un’euristica accurata. Anche se molte applicazioni della pianificazione nel mondo reale hanno utilizzato euristiche specifiche del dominio, risulta possibile (come vedremo nel Paragrafo 11.3) ricavare automaticamente buone euristiche indipendenti dal dominio; è proprio questo che rende praticabile la ricerca in avanti.

11.2.2 Ricerca all’indietro per pianificazione

Nella ricerca all’indietro, o **ricerca con regressione**, iniziamo dall’obiettivo e applichiamo le azioni all’indietro fino a trovare una sequenza di passi che raggiunge lo stato iniziale. A ogni passo consideriamo **azioni rilevanti** (a differenza della ricerca in avanti che considera azioni **applicabili**) e così si riduce notevolmente il fattore di ramificazione, soprattutto nei domini con molte azioni possibili.

Un’azione rilevante è un’azione con un effetto che si **unifica** con uno dei letterali obiettivo, ma priva di effetti che neghino qualsiasi parte dell’obiettivo stesso. Per esempio, con l’obiettivo $\neg Povero \wedge Famoso$, un’azione con l’unico effetto *Famoso* sarebbe rilevante, mentre un’azione con l’effetto *Povero* \wedge *Famoso* non sarebbe considerata rilevante, anche se potrebbe essere usata in qualche punto del piano (per stabilire *Famoso*), non può apparire nel punto *attuale* del piano perché allora *Povero* apparirebbe nello stato finale.

Che cosa significa applicare un’azione all’indietro? Dato un obiettivo *g* e un’azione *a*, la **regressione** da *g* su *a* fornisce una descrizione di stato *g'* i cui letterali positivo e negativo sono dati da:

$$\begin{aligned} POS(g') &= (POS(g) - ADD(a)) \cup POS(Precond(a)) \\ NEG(g') &= (NEG(g) - DEL(a)) \cup NEG(Precond(a)). \end{aligned}$$

ricerca con regressione
azione rilevante

regressione

In pratica, è necessario imporre che le precondizioni valessero prima, altrimenti non sarebbe stato possibile eseguire l'azione, mentre non è necessario che anche i letterali positivi/negativi che sono stati aggiunti/eliminati dall'azione fossero veri prima.

Queste equazioni sono semplici per i letterali ground, ma quando ci sono variabili in g e a è necessario procedere con attenzione. Per esempio, supponete che l'obiettivo sia conseguire una specifica merce a SFO : $\text{Posizione}(C_2, SFO)$. Lo schema di azione *Scarica* ha l'effetto $\text{Posizione}(c, a)$; quando lo unifichiamo con l'obiettivo, otteniamo la sostituzione $\{c/C_2, a/SFO\}$; applicando tale sostituzione allo schema otteniamo un nuovo schema che cattura l'idea di usare qualsiasi aereo che si trovi in SFO :

$Azione(\text{Scarica}(C_2, p', SFO))$,

PRECOND: $\text{In}(C_2, p') \wedge \text{Posizione}(p', SFO) \wedge \text{Merci}(C_2) \wedge \text{Aereo}(p') \wedge \text{Aeroporto}(SFO)$

EFFETTO: $\text{Posizione}(C_2, SFO) \wedge \neg \text{In}(C_2, p')$.

In questo caso abbiamo sostituito p con una nuova variabile denominata p' , con un'operazione di **standardizzazione** dei nomi delle variabili che ha lo scopo di evitare eventuali conflitti tra variabili diverse che potrebbero avere lo stesso nome (cfr. Paragrafo 9.2.1). La descrizione dello stato regredito ci fornisce un nuovo obiettivo:

$g' = \text{In}(C_2, p') \wedge \text{Posizione}(p', SFO) \wedge \text{Merci}(C_2) \wedge \text{Aereo}(p') \wedge \text{Aeroporto}(SFO)$.

Per fare un altro esempio, consideriamo l'obiettivo di possedere un libro con uno specifico codice ISBN: $\text{Possiede}(9780134610993)$. Dati 10.000 miliardi di ISBN a 13 cifre e il singolo schema di azione:

$A = Azione(\text{Acquista}(i))$, PRECOND: $\text{ISBN}(i)$, EFFETTO: $\text{Possiede}(i)$,

una ricerca in avanti senza euristica dovrebbe iniziare enumerando i 10.000 miliardi di azioni ground *Acquista*. Con una ricerca all'indietro, invece, unificheremmo l'obiettivo $\text{Possiede}(9780134610993)$ con l'effetto $\text{Possiede}(i')$, ottenendo la sostituzione $\theta = \{i'/9780134610993\}$. Poi effettueremmo la regressione sull'azione $\text{SUBST}(\theta, A')$ per ottenere la descrizione di stato predecessore $\text{ISBN}(9780134610993)$. Questa fa parte dello stato iniziale, perciò abbiamo una soluzione e abbiamo terminato, dopo aver considerato una sola azione e non 10.000 miliardi.

In termini più formali, ipotizziamo una descrizione di un obiettivo g che contiene un letterale obiettivo g_i e uno schema di azione A . Se A ha un letterale effetto e'_j , dove $\text{UNIFY}(g_i, e'_j) = \theta$ e dove definiamo $A' = \text{SUBST}(\theta, A)$, e se non esiste un effetto in A' che sia la negazione di un letterale in g , allora A' è un'azione rilevante rispetto a g .

Nella maggior parte dei domini di problemi la ricerca all'indietro mantiene il fattore di ramificazione più basso rispetto alla ricerca in avanti. Tuttavia, il fatto che la ricerca all'indietro utilizzi stati con variabili anziché stati ground può complicare la definizione di una buona euristica. È questo il motivo principale per cui la maggior parte dei sistemi attuali preferisce la ricerca in avanti.

11.2.3 Pianificazione come soddisfacibilità booleana

Nel Paragrafo 7.7.4 abbiamo visto come, riscrivendo opportunamente alcuni assiomi, si possa trasformare un problema del mondo del wumpus in un problema di soddisfacibilità in logica proposizionale che potrebbe essere gestito da un risolutore efficiente per la soddisfacibilità. I pianificatori basati su SAT come SATPLAN operano traducendo una descrizione di problema PDDL in forma proposizionale. La traduzione avviene in una serie di passaggi descritti di seguito.

- Proposizionalizzare le azioni: per ogni schema di azione, formare proposizioni ground sostituendo ognuna delle variabili con delle costanti. Anziché un singolo schema *Scarica*(c, p, a) dovremmo avere proposizioni separate che rappresentano le azioni per

ogni combinazione di merce, aereo e aeroporto (nel seguito indicati con pedici) e per ogni passo temporale (nel seguito indicato in apice).

- Aggiungere assiomi di esclusione che affermano che due azioni non possono mai avvenire nello stesso tempo, per esempio $\neg(VolaP_1SFOJFK^1 \wedge VolaP_1SFOBUH^1)$.
 - Aggiungere assiomi di precondizione: per ogni azione ground A , aggiungere l'assioma $A^t \Rightarrow PRE(A)^t$, ovvero, se un'azione è eseguita al tempo t , allora le precondizioni dovevano essere vere. Per esempio, $VolaP_1SFOJFK^1 \Rightarrow Posizione(P_1, SFO) \wedge Aereo(P_1) \wedge Aeroporto(SFO) \wedge Aeroporto(JFK)$.
 - Definire lo stato iniziale: asserire F^0 per ogni fluente F nello stato iniziale del problema e $\neg F^0$ per ogni fluente non menzionato nello stato iniziale.
 - Proposizionalizzare l'obiettivo, che diventa una disgiunzione su tutte le sue istanze ground, dove le variabili sono sostituite da costanti. Per esempio, l'obiettivo di avere il blocco A sopra un altro blocco, $Sopra(A, x) \wedge Blocco(x)$ in un mondo con oggetti A, B e C sarebbe sostituito dall'obiettivo:
- $$(Sopra(A, A) \wedge Blocco(A)) \vee (Sopra(A, B) \wedge Blocco(B)) \vee (Sopra(A, C) \wedge Blocco(C)).$$
- Aggiungere assiomi di stato successore: per ogni fluente F , aggiungere un assioma della forma:

$$F^{t+1} \Leftrightarrow AzioneCausaF^t \vee (F^t \wedge \neg AzioneCausaNotF^t),$$

dove $AzioneCausaF$ è una disgiunzione di tutte le azioni ground che aggiungono F e $AzioneCausaNotF$ è una disgiunzione di tutte le azioni ground che eliminano F .

La traduzione risultante è generalmente molto più lunga della formulazione PDDL originale, ma l'efficienza dei moderni risolutori SAT spesso è più che sufficiente per gestirla.

11.2.4 Altri approcci di pianificazione classica

I tre approcci descritti finora non sono i soli che sono stati tentati nei 50 anni di storia della pianificazione automatica. Di seguito ne descriviamo brevemente alcuni altri.

Un approccio denominato Graphplan utilizza una speciale struttura dati, detta **grafo di pianificazione**, per codificare vincoli sulle relazioni tra le azioni e le loro precondizioni ed effetti, e su quali relazioni sono mutuamente esclusive.

Il **calcolo delle situazioni** è un metodo per descrivere problemi di pianificazione in logica del primo ordine. Utilizza assiomi di stato successore proprio come fa SATPLAN, ma la logica del primo ordine consente una maggiore flessibilità e assiomi più sintetici. Tale approccio ha complessivamente contribuito alla comprensione teorica della pianificazione, ma non ha avuto grande impatto nelle applicazioni pratiche, forse perché i dimostratori del primo ordine non sono ben sviluppati come lo sono i programmi di soddisfacibilità proposizionale.

È possibile codificare un problema di pianificazione limitato (per esempio il problema di trovare un piano di lunghezza k) come **problema di soddisfacimento di vincoli** o CSP (*constraint satisfaction problem*). La codifica è simile a quella utilizzata per i problemi SAT (Paragrafo 11.2.3), con una importante semplificazione: a ogni passo temporale ci serve soltanto una singola variabile $Azione^t$ il cui dominio è l'insieme delle azioni possibili, non ci serve più una variabile per ogni azione e non abbiamo bisogno degli assiomi di esclusione di azioni.

Tutti gli approcci visti finora costruiscono piani *completamente ordinati* costituiti da sequenze strettamente lineari di azioni. Tuttavia, se in un problema di trasporto merci per via aerea ci sono 30 pacchi di merce da caricare su un aereo e 50 pacchi da caricare in un altro, sembra inutile stabilire uno specifico ordinamento lineare delle 80 azioni di caricamento.

In un altro approccio, la **pianificazione con ordinamento parziale**, un piano è rappresentato come grafo anziché come sequenza lineare: ogni azione è un nodo del grafo, e per ogni

grafo di pianificazione

calcolo delle situazioni

pianificazione con ordinamento parziale

precondizione dell'azione esiste un collegamento da un'altra azione (o dallo stato iniziale) che indica che l'azione predecessore stabilisce la precondizione. Potremmo avere, quindi, un piano parzialmente ordinato che afferma che le azioni *Rimuovi(Scorta, Bagagliaio)* e *Rimuovi(Bucata, Asse)* devono precedere *Monta(Scorta, Asse)*, senza però dire quale delle due azioni *Rimuovi* debba precedere l'altra. La ricerca è effettuata nello spazio dei piani e non degli stati del mondo, inserendo azioni per soddisfare condizioni.

Negli anni 1980 e 1990 la pianificazione con ordinamento parziale era considerata il modo migliore per gestire problemi di pianificazione con sottoproblemi indipendenti. Nel 2000 i pianificatori con ricerca in avanti avevano sviluppato eccellenti euristiche che consentivano di scoprire in modo efficiente i sottoproblemi indipendenti per cui era stata progettata la pianificazione con ordinamento parziale, inoltre SATPLAN era in grado di sfruttare la legge di Moore: una proposizionalizzazione che nel 1980 appariva troppo grande per poter essere affrontata, oggi appare piccola, perché i computer di oggi hanno una quantità di memoria 10.000 volte maggiore rispetto ad allora. Di conseguenza, i pianificatori con ordinamento parziale non sono competitivi su problemi di pianificazione classica completamente automatica.

La pianificazione con ordinamento parziale è ancora usata frequentemente in domini in cui è importante che gli uomini possano comprendere i piani. Per esempio, i piani operativi per veicoli spaziali e robot di esplorazione di Marte sono generati da sistemi di pianificazione con ordinamento parziale e vengono poi controllati da operatori umani prima di essere inviati ai veicoli per l'esecuzione. L'approccio con raffinamento dei piani facilita agli uomini il compito di comprendere ciò che fanno gli algoritmi di pianificazione e verificare che siano corretti prima della loro esecuzione.

11.3 Euristiche per la pianificazione

Né la ricerca in avanti né quella all'indietro sono efficienti senza una buona funzione eistica. Ricordiamo dal Capitolo 3 che una funzione eistica $h(s)$ stima la distanza da uno stato s all'obiettivo, e che, se siamo in grado di derivare un'eistica **ammissibile** (che non fornisca una sovrastima) per tale distanza, possiamo utilizzare la ricerca A* per trovare soluzioni ottime.

Per definizione, non è possibile analizzare uno stato atomico, perciò un analista (solitamente umano) dovrebbe essere assai ingegnoso per definire buone euristiche specifiche del dominio per problemi di ricerca con stati atomici. La pianificazione però utilizza una rappresentazione fattorizzata per stati e azioni; questo consente di definire buone euristiche indipendenti dal dominio.

Ricordiamo che un'eistica ammissibile può essere ricavata definendo un **problema rilassato** che sia più facile da risolvere. Il costo esatto di una soluzione per il problema più facile diviene quindi l'eistica per il problema originale. Un problema di ricerca è un grafo in cui i nodi sono stati e gli archi sono azioni. Il problema consiste nel trovare un cammino che collega lo stato iniziale a uno stato obiettivo. Sono due i modi principali per rilassare questo problema rendendolo più facile: aggiungendo più archi al grafo, e facilitando così la ricerca di un cammino, oppure raggruppando più nodi tra loro a formare un'astrazione dello spazio degli stati che abbia meno stati e consenta quindi una ricerca più facile.

Esaminiamo prima le euristiche che aggiungono archi al grafo. Quella forse più semplice è l'**eistica che ignora le precondizioni**, che rimuove tutte le precondizioni dalle azioni. Ogni azione diventa applicabile in ogni stato, e ogni singolo fluente obiettivo può essere raggiunto in un solo passo (se esistono azioni applicabili, altrimenti il problema è impossibile). Questo quasi implica che il numero di passi richiesti per risolvere il problema rilassato è il numero di obiettivi insoddisfatti – quasi ma non completamente, perché (1) alcune azioni potrebbero raggiungere obiettivi multipli e (2) alcune azioni potrebbero annullare gli effetti di altre.

Per molti problemi si ottiene un'euristica accurata considerando (1) e ignorando (2). Per prima cosa rilassiamo le azioni rimuovendo tutte le precondizioni e tutti gli effetti eccetto quelli che sono letterali dell'obiettivo. Poi contiamo il numero minimo di azioni richieste perché l'unione dei loro effetti soddisfi l'obiettivo. Questa è un'istanza del **problema set-cover** (copertura di un insieme), che presenta una difficoltà: il problema set-cover è NP-difficile. Fortunatamente un semplice algoritmo greedy offre la garanzia di restituire una copertura con dimensione entro un fattore $\log n$ della dimensione della vera copertura minima, dove n è il numero di letterali nell'obiettivo. Sfortunatamente, l'algoritmo greedy perde la garanzia di ammissibilità.

problema set-cover

È anche possibile ignorare soltanto alcune precondizioni selezionate. Consideriamo il rompicapo a 8 tasselli o a 15 tasselli del Paragrafo 3.2. Potremmo codificarlo come un problema di pianificazione che riguarda i tasselli con un singolo schema *Scorre*:

Azione(*Scorre*(t, s_1, s_2),

PRECOND: $Su(t, s_1) \wedge Tassello(t) \wedge Vuoto(s_2) \wedge Adiacente(s_1, s_2)$

EFFETTO: $Su(t, s_2) \wedge Vuoto(s_1) \wedge Su(t, s_1) \wedge \neg Vuoto(s_2)$.

Come abbiamo visto nel Paragrafo 3.6, se rimuoviamo le precondizioni $Vuoto(s_2) \wedge Adiacente(s_1, s_2)$, allora ogni tassello può muoversi in una sola azione fino a qualsiasi spazio e otteniamo l'euristica del numero di tasselli fuori posto. Se rimuoviamo soltanto la precondizione $Vuoto(s_2)$ otteniamo l'euristica della distanza Manhattan. È facile vedere che queste euristiche potrebbero essere derivate automaticamente dalla descrizione dello schema di azione. La facilità con cui si possono manipolare gli schemi di azione è il grande vantaggio della rappresentazione fattorizzata dei problemi di pianificazione, rispetto alla rappresentazione atomica dei problemi di ricerca.

Un'altra possibilità è quella dell'euristica che **ignora le liste di eliminazioni**. Ipotizziamo per un momento che tutti gli obiettivi e le precondizioni contengano soltanto letterali positivi.² Vogliamo creare una versione rilassata del problema originale che sia più facile da risolvere e in cui la lunghezza della soluzione servirà come buona euristica. Possiamo farlo rimuovendo le liste di eliminazioni da tutte le azioni (cioè rimuovendo tutti i letterali negativi dagli effetti). Ciò consente di progredire in maniera monotona verso l'obiettivo: nessuna azione annullerà mai il progresso fatto da un'altra. Risulta che è ancora NP-difficile trovare la soluzione ottima di questo problema rilassato, ma una soluzione approssimata si può trovare in tempo polinomiale mediante hill climbing.

ignorare le liste di eliminazioni

La Figura 11.6 illustra parte dello spazio degli stati per due problemi di pianificazione utilizzando l'euristica che ignora le liste di eliminazioni. I punti rappresentano stati e gli archi azioni; l'altezza di ciascun punto sopra il piano inferiore rappresenta il valore dell'euristica. Gli stati sul piano inferiore sono soluzioni. In entrambi questi problemi, c'è un ampio cammino che porta all'obiettivo. Non ci sono vicoli ciechi, perciò non serve il backtracking; una semplice ricerca hill climbing troverà facilmente una soluzione (anche se potrebbe non essere una soluzione ottima).

11.3.1 Potatura indipendente dal dominio

Con le rappresentazioni fattorizzate appare ovvio che molti stati sono soltanto varianti di altri stati. Per esempio, supponiamo di avere una dozzina di blocchi su un tavolo e che l'obiettivo sia quello di avere il blocco *A* sopra una torre di tre blocchi. Il primo passo di una

² Molti problemi sono scritti con questa convenzione. Per quelli che non lo sono, sostituite ogni letterale negativo $\neg P$ in un obiettivo o precondizione con un nuovo letterale positivo P' e modificate lo stato iniziale e gli effetti delle azioni in modo conseguente.

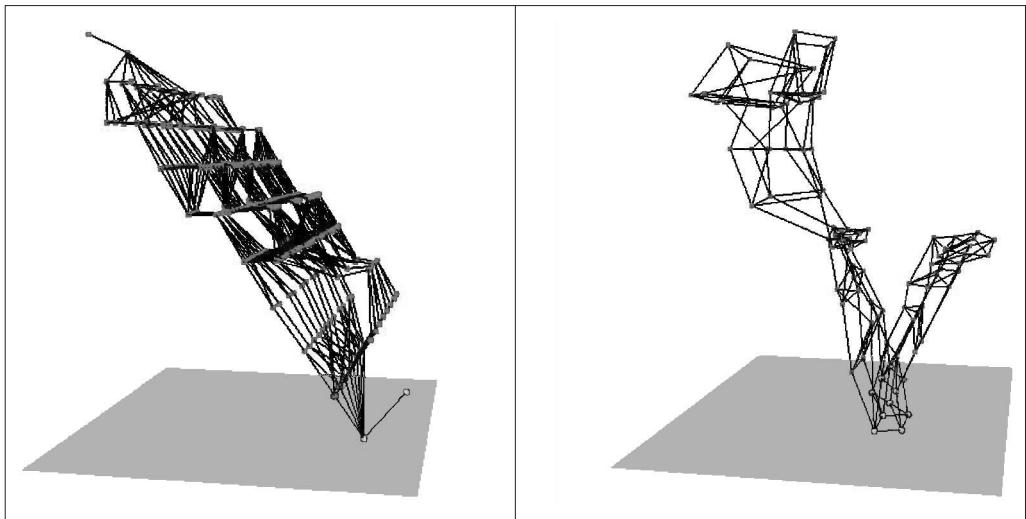


Figura 11.6 Due spazi degli stati per problemi di pianificazione con euristica che ignora le liste di eliminazioni. L'altezza sopra il piano inferiore è il valore dell'euristica per uno stato; gli stati sul piano inferiore sono obiettivi. Non ci sono minimi locali, perciò la ricerca dell'obiettivo è semplice. Da Hoffmann (2005).

riduzione
simmetrica

azione preferita

sotto-obiettivi
serializzabili

soluzione consiste nel posizionare un blocco x sopra un blocco y ($x, y \in A$ devono essere tutti diversi), poi basta posizionare A sopra x e abbiamo terminato. Ci sono 11 scelte disponibili per x , e una volta effettuata la scelta di x , rimangono 10 scelte per y , quindi in totale ci sono 110 stati da considerare. Tutti questi stati sono però simmetrici: scegliere uno piuttosto che l'altro non fa differenza, quindi un pianificatore dovrebbe considerarne soltanto uno. Questo processo si chiama **riduzione simmetrica**: andiamo a potare (ignorandoli) tutti i rami simmetrici dell'albero di ricerca eccetto uno. Per molti domini questo fa la differenza tra un problema intrattabile e una risoluzione efficiente.

Un'altra possibilità è quella di attuare una potatura in avanti, accettando il rischio di potare anche una soluzione ottima, per focalizzare la ricerca sui rami più promettenti. Possiamo definire un'**azione preferita** come segue: per prima cosa definiamo una versione rilassata del problema e risolviamola per ottenere un **piano rilassato**; a questo punto un'azione preferita è un passo del piano rilassato, oppure ottiene una precondizione di tale piano.

A volte è possibile risolvere un problema in modo efficiente rendendosi conto che è possibile eliminare le interazioni negative. Diciamo che un problema ha **sotto-obiettivi serializzabili** se esiste un ordinamento di sotto-obiettivi tale che il pianificatore può raggiungerli in tale ordine senza dover annullare alcuno dei sotto-obiettivi raggiunti precedentemente. Per esempio, nel mondo dei blocchi, se l'obiettivo è costruire una torre (per esempio con il blocco A sopra B che a sua volta è sopra il blocco C , che a sua volta è sopra il *Tavolo*, come nella precedente Figura 11.3), allora i sotto-obiettivi sono serializzabili dal basso verso l'alto: se raggiungiamo prima l'obiettivo di C sopra *Tavolo*, non avremo mai la necessità di annullarlo mentre andiamo a raggiungere gli altri sotto-obiettivi. Un pianificatore che adotti questo approccio “dal basso verso l'alto” può risolvere qualsiasi problema nel mondo dei blocchi senza ricorrere al backtracking (anche se non è detto che trovi sempre il piano più breve). Per fare un altro esempio, se c'è una stanza con n interruttori della luce, ognuno dei quali controlla una luce separata, e l'obiettivo è che le luci siano tutte accese, non dobbiamo considerare permutazioni dell'ordinamento, potremmo limitarci arbitrariamente ai piani che attivano gli interruttori della luce in ordine ascendente, per esempio.

Per il pianificatore Remote Agent che ha controllato la sonda spaziale Deep Space One della NASA, si è determinato che le proposizioni coinvolte nel comando di una sonda spa-

ziale sono serializzabili. Ciò non desta particolare sorpresa, dato che una sonda spaziale è *progettata* dai tecnici per la massima facilità di controllo possibile (dati gli altri vincoli esistenti). Sfruttando il vantaggio dell'ordinamento di obiettivi serializzabili, Remote Agent ha potuto così eliminare gran parte delle ricerche, il che gli ha consentito di raggiungere una velocità sufficiente per controllare la sonda in tempo reale, cosa che in precedenza era considerata impossibile.

11.3.2 Astrazione di stato nella pianificazione

Un problema rilassato ci consegna un problema di pianificazione semplificato per il solo calcolo del valore della funzione euristica. Molti problemi di pianificazione hanno 10^{100} stati o anche più, e il rilassamento delle *azioni* non serve a ridurre il numero degli stati, il che significa che potrebbe essere ancora troppo costoso calcolare l'euristica. Di conseguenza, ora consideriamo rilassamenti che riducono il numero degli stati formando una **astrazione di stato**, ovvero una corrispondenza molti a uno dagli stati nella rappresentazione originale (ground) del problema alla rappresentazione astratta.

La forma più semplice di astrazione di stato consiste nell'ignorare alcuni fluenti. Per esempio, consideriamo un problema di trasporto aereo con 10 aeroporti, 50 aerei e 200 pacchi di merce. Ogni aereo può trovarsi in uno dei 10 aeroporti e ogni pacco di merce può trovarsi in uno degli aerei oppure scaricato presso uno degli aeroporti. Gli stati sono quindi $10^{50} \times (50+10)^{200} \approx 10^{405}$. Ora consideriamo un particolare problema in tale dominio, in cui tutti i pacchi di merce si trovano in soltanto 5 degli aeroporti, e tutti i pacchi che si trovano presso un dato aeroporto hanno la stessa destinazione. Un'utile astrazione del problema consiste nello scartare tutti i fluenti *Posizione* tranne quelli che riguardano un solo aereo e un solo pacco di merce presso ognuno dei 5 aeroporti. Ora gli stati sono soltanto $10^5 \times (5+10)^5 \approx 10^{11}$. Una soluzione in questo spazio degli stati astratto sarà più breve di una soluzione nello spazio originale (e quindi sarà un'euristica ammissibile), e la soluzione astratta si estende facilmente a una soluzione del problema originale (aggiungendo azioni *Carica* e *Scarica*).

Un concetto chiave nella definizione di euristiche è la **scomposizione**: dividere un problema in parti, risolvere ogni parte in modo *indipendente* e poi rimettere insieme le parti. L'**ipotesi dell'indipendenza dei sotto-obiettivi**, secondo la quale il costo di risolvere una congiunzione di sotto-obiettivi è approssimato dalla somma dei costi di risolvere ciascun sotto-obiettivo in modo indipendente, può essere ottimistica o pessimistica. È ottimistica quando ci sono interazioni negative tra i sottopiani per ciascun sotto-obiettivo (per esempio, quando un'azione di un sottopiano elimina un obiettivo raggiunto da un altro sottopiano). È pessimistica, e perciò inammissibile, quando i sottopiani contengono azioni ridondanti (per esempio, due azioni che potrebbero essere sostituite da una sola nel piano che li combina).

Supponiamo che l'obiettivo sia l'insieme di fluenti G , che dividiamo in sottoinsiemi disgiunti G_1, \dots, G_n . Troviamo poi i piani ottimi P_1, \dots, P_n che risolvono i rispettivi sotto-obiettivi. Qual è una stima del costo del piano per raggiungere l'intero G ? Possiamo considerare ogni $\text{COSTO}(P_i)$ come una stima euristica, e sappiamo che combinando le stime prendendo il loro valore massimo, otteniamo sempre un'euristica ammissibile. Perciò $\max_i \text{COSTO}(P_i)$ è ammissibile, e talvolta è esattamente corretta: può darsi perfino che P_1 raggiunga fortunosamente tutti i G_i . Nella maggior parte dei casi reali, tuttavia, la stima è troppo bassa. Potremmo allora sommare i costi? Per molti problemi sarebbe una stima ragionevole, ma non ammissibile. Il caso migliore si ha quando G_i e G_j sono indipendenti, nel senso che i piani per l'uno non possono ridurre il costo dei piani per l'altro. In quel caso, la stima $\text{COSTO}(P_i) + \text{COSTO}(P_j)$ è ammissibile e più accurata della stima data dal valore massimo.

È chiaro che formando astrazioni si ha la possibilità di ridurre la dimensione dello spazio di ricerca. Il trucco è di scegliere le astrazioni giuste e utilizzarle in un modo che renda il costo totale (definire un'astrazione, eseguire una ricerca astratta e far corrispondere l'astrazione

astrazione di stato

scomposizione

**indipendenza dei
sotto-obiettivi**

zione al problema originale) inferiore a quello di risolvere il problema originale. Le tecniche dei **database di pattern** descritte nel Paragrafo 3.6.3 possono essere utili, perché il costo di creare il database di pattern può essere ammortizzato su più istanze del problema.

Un sistema che utilizza euristiche efficienti è FF, o FASTFORWARD (Hoffmann, 2005), un programma di ricerca in avanti nello spazio degli stati che utilizza l'euristica che ignora le liste di eliminazioni, stimando l'euristica con l'aiuto di un grafo di pianificazione. FF utilizza poi la ricerca hill climbing (modificata in modo da tenere traccia del piano) con l'euristica per trovare una soluzione. L'algoritmo hill climbing di FF è particolare: evita i massimi locali eseguendo una ricerca in ampiezza a partire dallo stato corrente finché ne trova uno migliore; se fallisce, passa a una ricerca best-first greedy.

11.4 Pianificazione gerarchica

I metodi per la risoluzione di problemi e per la pianificazione dei precedenti capitoli operano tutti con un insieme fissato di azioni atomiche. Le azioni possono essere accodate e gli algoritmi possono generare soluzioni contenenti migliaia di azioni. Va tutto bene se stiamo pianificando una vacanza e le azioni sono del tipo “volare da San Francisco a Honolulu”, ma passando ad ambiti quali il controllo motorio con azioni quali “piega il ginocchio sinistro di 5 gradi” dovremmo accodare milioni o miliardi di azioni, non migliaia.

Per colmare questo gap occorre pianificare a livelli di astrazione più alti. Un piano di alto livello per la vacanza alle Hawaii potrebbe essere: “vai all'aeroporto di San Francisco; prendi il volo HA 11 per Honolulu; goditi la vacanza per due settimane; prendi il volo HA 12 per tornare a San Francisco; vai a casa”. Con un tale piano, l'azione “vai all'aeroporto di San Francisco” può essere considerata come un compito di pianificazione di per sé, con una soluzione quale “scegli un servizio di trasporto in auto; ordina un veicolo; vai fino all'aeroporto”. Ognuna di queste azioni, a loro volta, può essere scomposta ulteriormente, finché raggiungiamo azioni di basso livello per controllo motorio come premere un pulsante.

In questo esempio, pianificazione e azione sono inframmezzate; per esempio, probabilmente sarebbe bene differire il problema di pianificare un cammino da una piazzola di sosta al gate fino a quando si è scesi dal veicolo del servizio di trasporto. Quella particolare azione, quindi, rimarrà a un livello astratto prima della fase di esecuzione. Rimandiamo la trattazione di questo argomento al Paragrafo 11.5, mentre qui ci concentriamo sul concetto di **scomposizione gerarchica**, che pervade quasi tutti i tentativi di gestire la complessità. Per esempio, i software più complessi sono costruiti partendo da una gerarchia di procedure e classi; gli eserciti, i governi e le aziende hanno organizzazioni gerarchiche. Il principale beneficio di una struttura gerarchica è che, a ogni livello della gerarchia, un'attività computazionale, una missione militare o una funzione amministrativa si riducono a un *piccolo* numero di attività al livello immediatamente inferiore, cosicché il costo di trovare il modo migliore di organizzare tali attività risulta parimenti piccolo.

11.4.1 Azioni di alto livello

Il formalismo di base che adottiamo per comprendere la scomposizione gerarchica proviene dal campo della pianificazione con **reti gerarchiche** o HTN (*hierarchical task network*). Per ora assumiamo la completa osservabilità e la disponibilità di un insieme di azioni, ora chiamate **azioni primitive**, con schemi precondizione-effetto standard. Il concetto in più, fondamentale, è quello di **azione di alto livello** o HLA (*high-level action*), come per esempio, l'azione “vai all'aeroporto di San Francisco”. Ogni HLA ha uno o più possibili **raffinamenti**, in una sequenza di azioni, ognuna delle quali può essere una HLA o un'azione primitiva (che per definizione non ha raffinamenti). Per esempio, l'azione “vai all'aeroporto di San Francisco”, rappresentata formalmente come *Vai(Casa, SFO)*, potrebbe avere due possibili

scomposizione gerarchica

rete gerarchica

azione primitiva
azione di alto livello
raffinamento

```

Raffinamento(Vai(Casa, SFO)),
  PASSI: [Guida(Casa, ParcheggioLungaSostaSFO),
           Navetta(ParcheggioLungaSostaSFO, SFO)] )

Raffinamento(Vai(Casa, SFO)),
  PASSI: [Taxi(Casa, SFO)] )

Raffinamento(Naviga([a, b], [x, y])),
  PRECOND:  $a=x \wedge b=y$ 
  PASSI: [ ] )

Raffinamento(Naviga([a, b], [x, y])),
  PRECOND: Connesso([a, b], [a - 1, b])
  PASSI: [Sinistra,Naviga([a - 1, b], [x, y])] )

Raffinamento(Naviga([a, b], [x, y])),
  PRECOND: Connesso([a, b], [a + 1, b])
  PASSI: [Destra,Naviga([a + 1, b], [x, y])] )

...

```

Figura 11.7
Definizioni di possibili raffinamenti per due azioni di alto livello: andare all'aeroporto di San Francisco e muoversi nel mondo dell'aspirapolvere. Nell'ultimo caso, notate la natura ricorsiva dei raffinamenti e l'uso di precondizioni.

raffinamenti, come si vede nella Figura 11.7. La stessa figura mostra un raffinamento **ricorsivo** per la navigazione nel mondo dell'aspirapolvere: per arrivare a una destinazione, fai un passo, e poi vai alla destinazione.

Questi esempi mostrano che le azioni di alto livello e i loro raffinamenti incorporano conoscenza sulle modalità con cui eseguire delle azioni. Per esempio, i raffinamenti per *Vai(Casa, SFO)* dicono che per arrivare all'aeroporto si può guidare o utilizzare un servizio di trasporto; bere latte, sedersi e spostare il cavallo in e4 non sono attività da considerare.

Un raffinamento di una HLA che contiene soltanto azioni primitive si chiama **implementazione** dell'HLA. In un mondo a griglia, le sequenze *[Destra, Destra, Giù]* e *[Giù, Destra, Destra]* implementano entrambe l'HLA *Naviga([1, 3], [3, 2])*. Un'implementazione di un piano di alto livello (una sequenza di HLA) è la concatenazione di implementazioni di ciascuna HLA della sequenza. Date le definizioni precondizione-effetto di ciascuna azione primitiva, è facile determinare se una data implementazione di un piano di alto livello raggiunge l'obiettivo.

Possiamo dire, quindi, che *un piano di alto livello raggiunge l'obiettivo da un dato stato se almeno una delle sue implementazioni raggiunge l'obiettivo da tale stato*. In questa definizione, “almeno una” è fondamentale: non occorre che tutte le implementazioni raggiungano l'obiettivo, perché l'agente decide quale implementazione eseguirà. Quindi, l'insieme delle possibili implementazioni nella pianificazione HTN, ognuna delle quali può avere un risultato diverso, non coincide con l'insieme dei possibili risultati nella pianificazione non deterministica, dove viene richiesto che un piano funzioni per tutti i risultati perché l'agente non può scegliere il risultato; è la natura a farlo.

Il caso più semplice è una HLA che ha esattamente una implementazione. In tal caso possiamo calcolare le precondizioni e gli effetti dell'HLA da quelli dell'implementazione (cfr. Esercizio 11.HLAU) e poi trattare l'HLA esattamente come se fosse un'azione primitiva. Si può mostrare che con la giusta collezione di HLA la complessità temporale della ricerca non informata può passare da esponenziale a lineare nella profondità della soluzione, anche se trovare tale collezione di HLA potrebbe non essere compito facile in sé. Quando le HLA hanno più implementazioni possibili, ci sono due opzioni: una è quella di cercare un'implementazione che funziona, come nel Paragrafo 11.4.2; l'altra è di ragionare direttamente sulle



HLA, nonostante la molteplicità delle implementazioni, come è spiegato nel Paragrafo 11.4.3. Quest’ultimo metodo consente di derivare piani astratti che si possono dimostrare corretti, senza la necessità di considerare le loro implementazioni.

11.4.2 Ricerca di soluzioni primitive

La pianificazione HTN è spesso formulata con una singola azione di “alto livello” denominata *Agisci*, dove lo scopo è quello di trovare un’implementazione di *Agisci* che raggiunga l’obiettivo. Questo approccio è del tutto generale. Per esempio, i problemi di pianificazione classica possono essere definiti come segue: per ogni azione primitiva a_i , fornire un raffinamento di *Agisci* con i passi $[a_i, \text{Agisci}]$. Così si crea una definizione ricorsiva di *Agisci* che ci consente di aggiungere azioni. Ci serve però un modo per arrestare la ricorsione: possiamo farlo fornendo un ulteriore raffinamento di *Agisci*, con una lista vuota di passi e con una precondizione uguale all’obiettivo del problema. Questo dice che, se l’obiettivo è già raggiunto, allora l’implementazione giusta è quella di non fare nulla.

L’approccio porta a un semplice algoritmo: scegliere ripetutamente una HLA nel piano corrente e sostituirla con uno dei suoi raffinamenti, finché il piano raggiunge l’obiettivo. Una possibile implementazione basata sulla ricerca in ampiezza è mostrata nella Figura 11.8. I piani sono considerati in ordine di profondità di annidamento dei raffinamenti, anziché di numero di passi primitivi. È facile scrivere una versione dell’algoritmo con ricerca su grafo, o anche versioni con ricerca in profondità e ad approfondimento iterativo.

In sostanza, questa forma di ricerca gerarchica esplora lo spazio delle sequenze conformi alla conoscenza contenuta nella libreria HLA riguardo come si devono fare le cose. È possibile codificare una grande quantità di conoscenza, non solo nelle sequenze di azioni specificate in ciascun raffinamento, ma anche nelle precondizioni per i raffinamenti. Per alcuni domini, i pianificatori HTN sono stati in grado di generare piani enormi con pochissima ricerca. Per esempio, O-PLAN (Bell e Tate, 1985), che combina pianificazione HTN e scheduling, è stato utilizzato per sviluppare piani di produzione per Hitachi. Un problema tipico riguarda una linea di 350 diversi prodotti, 35 macchine di assemblaggio e oltre 2000 operazioni diverse. Il pianificatore genera uno schedule di 30 giorni con tre turni di 8 ore al giorno, che comprende milioni di passi. Un altro importante aspetto dei piani HTN è che essi sono,

```
function RICERCA-GERARCHICA(problema, gerarchia) returns una soluzione, o fallimento
  frontiera  $\leftarrow$  una coda FIFO con  $[\text{Agisci}]$  quale unico elemento
  while true do
    if VUOTA?(frontiera) then return fallimento
    piano  $\leftarrow$  POP(frontiera) // sceglie il piano meno profondo nella frontiera
    hla  $\leftarrow$  la prima HLA in piano, o null se non ce ne sono
    prefisso, suffisso  $\leftarrow$  le sottosequenze di azioni prima e dopo hla in piano
    risultato  $\leftarrow$  RISULTATO(problema.INIZIALE, prefisso)
    if hla è null then // perciò plan è primitivo e risultato è il suo risultato
      if problema.È-OBIETTIVO(risultato) then return piano
    else for each sequenza in RAFFINAMENTI(hla, risultato, gerarchia) do
      aggiungi CONCATENA(prefisso, sequenza, suffisso) a frontiera
```

Figura 11.8 Un’implementazione con ricerca in ampiezza della ricerca in avanti gerarchica per la pianificazione. Il piano iniziale fornito all’algoritmo è $[\text{Agisci}]$. La funzione RAFFINAMENTI restituisce un insieme di sequenze di azioni, una per ciascun raffinamento della HLA le cui precondizioni sono soddisfatte dallo stato specificato, *risultato*.

per definizione, strutturati in modo gerarchico; solitamente ciò li rende facilmente comprensibili agli umani.

I benefici computazionali della ricerca gerarchica si possono notare esaminando un caso idealizzato. Supponiamo che un problema di pianificazione abbia una soluzione con d azioni primitive. Per un pianificatore nello spazio degli stati con ricerca in avanti, non gerarchico, con b azioni ammesse per ogni stato, il costo è $O(b^d)$, come si è spiegato nel Capitolo 3. Per un pianificatore HTN, supponiamo di avere una struttura di raffinamento molto regolare: ogni azione non primitiva ha r possibili raffinamenti, ognuna in k azioni al livello inferiore successivo. Vogliamo sapere quanti diversi alberi di raffinamento esistono con questa struttura. Se ci sono d azioni al livello primitivo, il numero di livelli sotto la radice è $\log_k d$, perciò il numero di nodi di raffinamento interni è $1 + k + k^2 + \dots + k^{\log_k d - 1} = (d - 1)/(k - 1)$. Ogni nodo interno ha r possibili raffinamenti, perciò si possono costruire $r^{(d-1)/(k-1)}$ alberi di scomposizione regolari. Esaminando questa formula vediamo che mantenendo r piccolo e k grande potremmo ottenere enormi risparmi: prendiamo la k -esima radice del costo non gerarchico, se b e r sono comparabili. r piccolo e k grande significa una libreria di HLA con un piccolo numero di raffinamenti, ognuno dei quali porta a una lunga sequenza di azioni. Questo non è sempre possibile: lunghe sequenze di azioni che siano utilizzabili su un'ampia varietà di problemi sono estremamente rare.

La chiave della pianificazione HTN è una libreria di piani contenente metodi noti per implementare azioni di alto livello complesse. Un modo per costruire tale libreria consiste nell'*apprendere* i metodi dall'esperienza nella risoluzione dei problemi. Dopo la fondamentale esperienza di costruire un piano da zero, l'agente può salvare il piano nella libreria come metodo per implementare l'azione di alto livello definita dall'attività. In questo modo l'agente può diventare sempre più competente, nel tempo, costruendo nuovi metodi a partire da quelli già realizzati. Un importante aspetto di questo processo di apprendimento è la capacità di *generalizzare* i metodi costruiti, eliminando dettagli specifici dell'istanza di un problema (per esempio il nome del costruttore o l'indirizzo di un luogo) mantenendo soltanto gli elementi chiave del piano. Gli uomini non avrebbero potuto raggiungere i livelli di competenza che li contraddistinguono, senza un meccanismo simile.

11.4.3 Ricerca di soluzioni astratte

L'algoritmo di ricerca gerarchica del paragrafo precedente raffina le HLA fino a sequenze di azioni primitive per determinare se un piano funziona. Ciò va contro il buon senso: si dovrebbe essere in grado di determinare che il piano di alto livello a due HLA:

[*Guida(Casa, ParcheggioLungaSostaSFO), Navetta(ParcheggioLungaSostaSFO, SFO)*] conduce all'aeroporto senza dover determinare un percorso specifico, la scelta della piazzola al parcheggio e così via. La soluzione consiste nello scrivere descrizioni precondizione-effetto delle HLA, esattamente come facciamo per le azioni primitive. Dalle descrizioni dovrebbe essere semplice dimostrare che il piano di alto livello raggiunge l'obiettivo. Questo è il santo graal della pianificazione gerarchica, per così dire, perché se deriviamo un piano di alto livello che probabilmente raggiunge l'obiettivo, lavorando in un piccolo spazio di ricerca di azioni di alto livello, allora possiamo affidarci a esso e lavorare sul problema di raffinare ogni passo del piano. Questo ci fornisce la riduzione esponenziale che cerchiamo.

Affinché questo approccio funzioni, è necessario che ogni piano di alto livello che “afferma” di raggiungere l'obiettivo (in virtù delle descrizioni dei suoi passi) raggiunga effettivamente l'obiettivo nel senso definito precedentemente: deve avere almeno una implementazione che raggiunga l'obiettivo. Questa caratteristica è stata chiamata **proprietà di raffinamento discendente** per descrizioni HLA.

Scrivere descrizioni HLA che soddisfino la proprietà di raffinamento discendente è, in linea di principio, facile: purché le descrizioni siano *vere*, ogni piano di alto livello che affermi

proprietà di
raffinamento
discendente

di raggiungere l’obiettivo deve raggiungerlo effettivamente, altrimenti le descrizioni fanno false affermazioni sulle HLA. Abbiamo già visto come scrivere descrizioni vere per HLA che hanno esattamente una implementazione (Esercizio 11.HLAU); sorge un problema quando l’HLA ha implementazioni *multiple*: come possiamo descrivere gli effetti di un’azione che può essere implementata in molti modi diversi?

Una risposta sicura (almeno per problemi in cui tutte le precondizioni e tutti gli obiettivi sono positivi) è di includere soltanto gli effetti positivi raggiunti da *ogni* implementazione dell’HLA e gli effetti negativi di *qualsiasi* implementazione. Allora la proprietà di raffinamento discendente sarebbe soddisfatta. Sfortunatamente questa semantica per le HLA è troppo prudente. Consideriamo ancora l’HLA *Vai(Casa, SFO)*, che ha due raffinamenti, e supponiamo, per rimanere al nostro argomento, di avere un mondo semplice in cui si possa sempre guidare fino all’aeroporto e parcheggiare, ma prendere un taxi richieda la precondizione *Contanti*. In tal caso, *Vai(Casa, SFO)* non conduce sempre all’aeroporto; in particolare, fallisce se *Contanti* è falsa e perciò non possiamo asserire *Posizione(Agente, SFO)* come effetto dell’HLA. Questo però non ha senso; se l’agente non aveva *Contanti*, avrebbe guidato personalmente. Richiedere che un effetto valga per *ogni* implementazione equivale ad assumere che sarà *qualcun altro*, un avversario, a scegliere l’implementazione. I risultati multipli dell’HLA sono considerati esattamente come se l’HLA fosse un’azione **non deterministica**, come nel Paragrafo 4.3. Nel nostro caso, è l’agente stesso a scegliere l’implementazione.

non determinismo demoniaco
non determinismo angelico
insieme raggiungibile

Nella comunità dei linguaggi di programmazione si è coniato il termine **non determinismo demoniaco** per indicare il caso in cui è un avversario a fare le scelte, in contrasto con il **non determinismo angelico**, dove è l’agente stesso che fa le scelte. Prendiamo a prestito questo termine per definire una **semantica angelica** per le descrizioni delle HLA. Il concetto di base richiesto per comprendere la semantica angelica è l'**insieme raggiungibile** di una HLA: dato uno stato s , l’insieme raggiungibile per una HLA h , scritto $\text{RAGGIUNGIBILE}(s, h)$, è l’insieme di stati raggiungibili da una qualsiasi delle implementazioni di HLA. L’idea chiave è che l’agente può scegliere in *quale* elemento dell’insieme raggiungibile si ritroverà quando esegue l’HLA; quindi, una HLA con raffinamenti multipli è più “potente” della stessa HLA con meno raffinamenti. Possiamo anche definire l’insieme raggiungibile di una sequenza di HLA. Per esempio, l’insieme raggiungibile di una sequenza $[h_1, h_2]$ è l’unione di tutti gli insiemi raggiungibili ottenuti applicando h_2 in ogni stato nell’insieme raggiungibile di h_1 :

$$\text{RAGGIUNGIBILE}(s, [h_1, h_2]) = \bigcup_{s' \in \text{RAGGIUNGIBILE}(s, h_1)} \text{RAGGIUNGIBILE}(s', h_2).$$

Date queste definizioni, un piano di alto livello – una sequenza di HLA – raggiunge l’obiettivo se il suo insieme raggiungibile interseca l’insieme degli stati obiettivo (confrontate ciò con la condizione molto più forte per la semantica demoniaca, in cui ogni membro dell’insieme raggiungibile deve essere uno stato obiettivo). Viceversa, se l’insieme raggiungibile non interseca l’obiettivo, allora il piano non funziona. La Figura 11.9 illustra questi concetti.

Il concetto di insiemi raggiungibili porta direttamente a un algoritmo: cercare tra i piani di alto livello per trovarne uno il cui insieme raggiungibile intersechi l’obiettivo; una volta trovato, l’algoritmo può *affidarsi* a tale piano astratto, sapendo che funziona, e concentrarsi sul suo ulteriore raffinamento. Torneremo più avanti sugli aspetti algoritmici; per ora consideriamo come sono rappresentati gli effetti di una HLA – l’insieme raggiungibile per ogni possibile stato iniziale. Un’azione primitiva può impostare un fluente a *vero* o *falso*, oppure lasciarlo *invariato* (con effetti condizionali – cfr. il Paragrafo 11.5.1 – esiste una quarta possibilità: scambiare una variabile con la sua opposta).

Una HLA sotto semantica angelica può fare di più: può *controllare* il valore di un fluente, impostandolo a *vero* o *falso* in base all’implementazione scelta. Questo significa che una HLA può avere nove diversi effetti su un fluente: se inizialmente è *vero*, può mantenerlo sempre *vero*, renderlo sempre *falso*, o fare una scelta; se il fluente inizialmente è *falso*, può

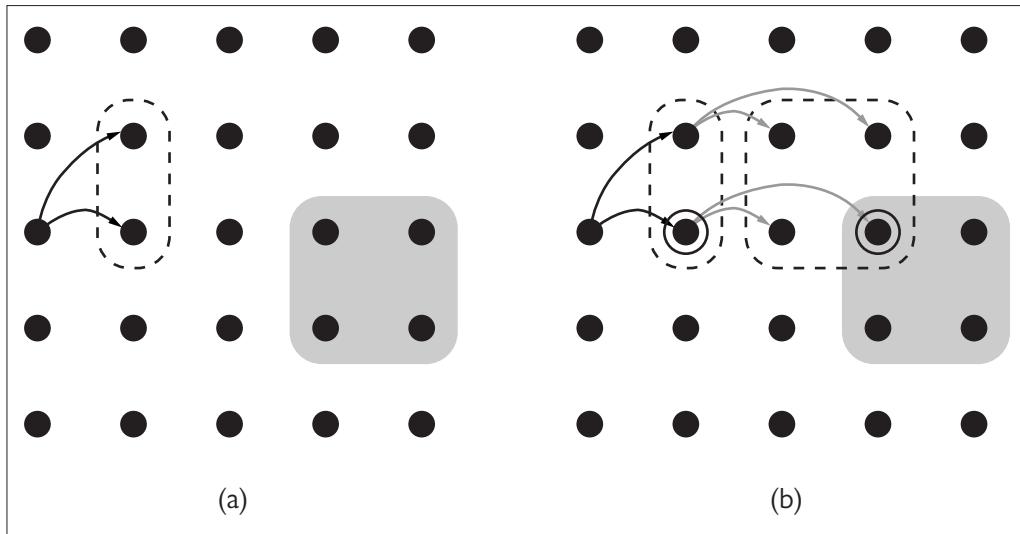


Figura 11.9 Esempi schematici di insiemi raggiungibili. L'insieme di stati obiettivo è colorato in grigio. Le frecce nere e grigie indicano possibili implementazioni di h_1 e h_2 , rispettivamente. (a) L'insieme raggiungibile di una HLA h_1 in uno stato s . (b) L'insieme raggiungibile per la sequenza $[h_1, h_2]$. Poiché questo insieme interseca l'insieme obiettivo, la sequenza raggiunge l'obiettivo.

mantenerlo sempre falso, renderlo sempre vero o fare una scelta; e le tre scelte per entrambi i casi possono essere combinate in modo arbitrario, generando nove scelte.

Dal punto di vista della notazione, questo pone dei problemi. Parleremo di aggiungere liste ed eliminare liste (anziché di fluenti veri/falsi) utilizzando il simbolo \sim per indicare “eventualmente, se l'agente sceglie così”. Perciò, l'effetto $\tilde{+} A$ significa “eventualmente aggiungi A ”, ovvero, lascia A invariato o rendilo vero. Analogamente, $\tilde{-} A$ significa “eventualmente elimina A ” e $\tilde{\pm} A$ significa “eventualmente aggiungi o elimina A ”. Per esempio, l'HLA *Vai(Casa, SFO)*, con i due raffinamenti mostrati nella Figura 11.7, eventualmente elimina *Contanti* (se l'agente decide di prendere un taxi), perciò dovrebbe avere l'effetto $\tilde{-} Contanti$. Vediamo quindi che le descrizioni delle HLA sono derivabili dalle descrizioni dei loro raffinamenti. Ora supponiamo di avere i seguenti schemi per le HLA h_1 e h_2 :

$$\begin{aligned} \text{Azione}(h_1, \text{PRECOND: } \neg A, \text{EFFETTO: } A \wedge \tilde{-} B), \\ \text{Azione}(h_2, \text{PRECOND: } \neg B, \text{EFFETTO: } \tilde{+} A \wedge \tilde{\pm} C). \end{aligned}$$

Ciò significa che h_1 aggiunge A ed eventualmente elimina B , mentre h_2 eventualmente aggiunge A e ha pieno controllo su C . Ora, se soltanto B è vera nello stato iniziale e l'obiettivo è $A \wedge C$, allora la sequenza $[h_1, h_2]$ raggiunge l'obiettivo: sceglio un'implementazione di h_1 che rende B falsa, poi sceglio un'implementazione di h_2 che rende A vera e C vera.

Nella discussione precedente si assume che gli effetti di una HLA – l'insieme raggiungibile per ogni stato iniziale dato – possano essere descritti in modo esatto descrivendo l'effetto su ciascun fluente. Sarebbe bello se fosse sempre così, ma in molti casi possiamo soltanto approssimare gli effetti, perché una HLA potrebbe avere infinite implementazioni e potrebbe produrre insiemi raggiungibili arbitrariamente contorti, un po' come nel problema con stato-credenza contorto illustrato nella Figura 7.21 del Capitolo 7. Per esempio, abbiamo detto che *Vai(Casa, SFO)* eventualmente elimina *Contanti*; inoltre, eventualmente aggiunge *Posizione(Auto, ParcheggioLungaSostaSFO)*; ma non può fare entrambe le cose, e in effetti ne deve fare una e una sola. Come per gli stati-credenza, potremmo avere la necessità di scrivere descrizioni *approssimate*. Utilizzeremo due tipi di approssimazione: una **descrizione ottimistica** RAGGIUNGIBILE⁺(s, h) di una HLA h potrebbe sovrastimare l'insieme raggiungibile.

descrizione ottimistica

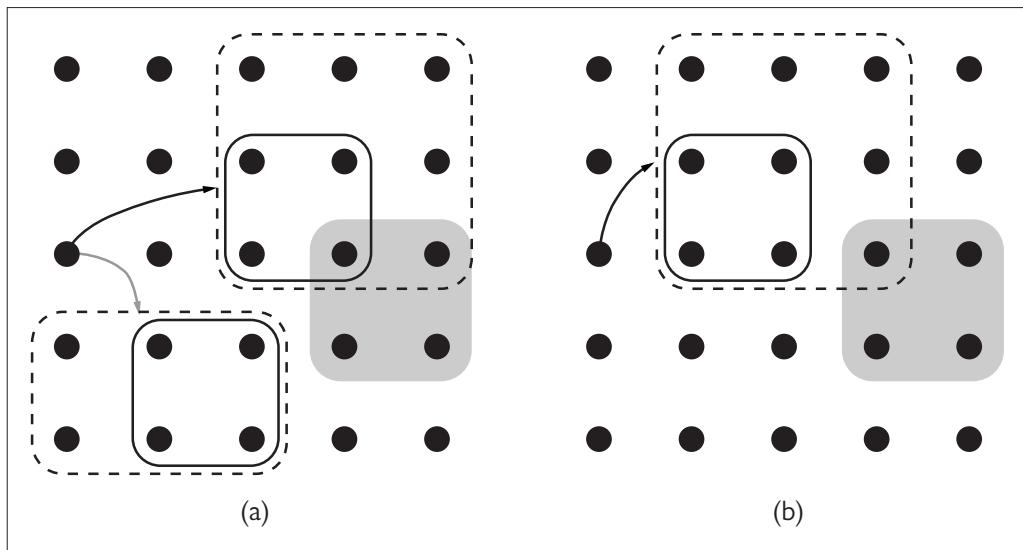


Figura 11.10 Raggiungimento dell'obiettivo per piani di alto livello con descrizioni approssimate. L'insieme di stati obiettivo è colorato in grigio. Per ogni piano sono mostrati l'insieme raggiungibile ottimistico (linee continue) e quello pessimistico (linee tratteggiate). (a) Il piano indicato dalla freccia nera raggiunge certamente l'obiettivo, mentre quello indicato dalla freccia grigia certamente non lo raggiunge. (b) Un piano che eventualmente raggiunge l'obiettivo (l'insieme raggiungibile ottimistico interseca l'obiettivo) ma non lo raggiunge necessariamente (l'insieme raggiungibile pessimistico non interseca l'obiettivo). Il piano dovrebbe essere ulteriormente raffinato per determinare se raggiunge realmente l'obiettivo.

descrizione pessimistica

gibile, mentre una **descrizione pessimistica** $\text{RAGGIUNGIBILE}^-(s, h)$ potrebbe sottostimarla. Abbiamo quindi:

$$\text{RAGGIUNGIBILE}^-(s, h) \subseteq \text{RAGGIUNGIBILE}(s, h) \subseteq \text{RAGGIUNGIBILE}^+(s, h).$$

Per esempio, una descrizione ottimistica di *Vai(Casa, SFO)* dice che eventualmente elimina *Contanti* ed eventualmente aggiunge *Posizione(Auto, ParcheggioLungaSostaSFO)*. Un altro buon esempio si può trovare nel rompicapo a 8 tasselli, in cui metà degli stati sono irraggiungibili da ogni stato dato (cfr. Esercizio 11.PART): la descrizione ottimistica di *Agisci* potrebbe includere l'intero spazio degli spazi, poiché l'insieme raggiungibile esatto è piuttosto contorto.

Con descrizioni approssimate, il test per verificare se un piano raggiunge l'obiettivo deve essere leggermente modificato. Se l'insieme raggiungibile ottimistico per il piano non interseca l'obiettivo, il piano non funziona; se l'insieme raggiungibile pessimistico interseca l'obiettivo, il piano funziona (Figura 11.10(a)). Con descrizioni esatte, un piano può funzionare o meno, ma con descrizioni approssimate esiste un terreno di mezzo: se l'insieme ottimistico interseca l'obiettivo, ma l'insieme pessimistico non lo interseca, non possiamo dire se il piano funziona (Figura 11.10(b)). Quando si verifica questa circostanza, l'incertezza può essere risolta raffinando il piano. Questa è una situazione molto comune nel ragionamento umano; per esempio, nel pianificare la vacanza di due settimane alle Hawaii citata precedentemente, si potrebbe proporre di passarle in sette isole, due giorni in ciascuna. Per prudenza questo ambizioso piano dovrebbe essere raffinato aggiungendo dettagli sui trasporti da un'isola all'altra.

Un algoritmo per la pianificazione gerarchica con descrizioni angeliche approssimate è mostrato nella Figura 11.11. Per semplicità abbiamo mantenuto lo stesso schema complesso utilizzato precedentemente nella Figura 11.8, cioè una ricerca in ampiezza nello spazio dei raffinamenti. Come abbiamo appena spiegato, l'algoritmo può individuare i piani che funzioneranno e quelli che non funzioneranno verificando le intersezioni degli insiemi rag-

```

function RICERCA-ANGELICA(problema, gerarchia, pianoIniziale) returns soluzione o fallimento
  frontiera  $\leftarrow$  una coda FIFO con pianoIniziale quale unico elemento
  while true do
    if VUOTA?(frontiera) then return fallimento
    piano  $\leftarrow$  POP(frontiera) // sceglie il nodo meno profondo in frontiera
    if RAGGIUNGIBILE+(problema.STATO-INIZIALE, piano) interseca problema.OBIETTIVO then
      if piano è primitivo then return piano // RAGGIUNGIBILE+ è esatto per piani primitivi
      garantito  $\leftarrow$  RAGGIUNGIBILE-(problema.STATO-INIZIALE, piano)  $\cap$  problema.OBIETTIVO
      if garantito  $\neq \{\}$  e PROGRESSO(piano, pianoIniziale) then
        statoFinale  $\leftarrow$  qualsiasi elemento di garantito
        return SCOMPONI(gerarchia, problema.STATO-INIZIALE, piano, statoFinale)
      hla  $\leftarrow$  una HLA in piano
      prefisso,suffisso  $\leftarrow$  le sottosequenze di azioni prima e dopo hla in piano
      risultato  $\leftarrow$  RISULTATO(problema.STATO-INIZIALE, prefisso)
      for each sequenza in RAFFINAMENTI(hla, risultato, gerarchia) do
        aggiungi CONCATENA(prefisso, sequenza, suffisso) a frontiera

function SCOMPONI(gerarchia, s0, piano, sf) returns una soluzione
  soluzione  $\leftarrow$  un piano vuoto
  while piano non è vuoto do
    azione  $\leftarrow$  RIMUOVI-ULTIMO(piano)
    si  $\leftarrow$  uno stato in RAGGIUNGIBILE-(s0, piano) tale che sf  $\in$  RAGGIUNGIBILE-(si, azione)
    problema  $\leftarrow$  un problema con STATO-INIZIALE = si e OBIETTIVO = sf
    soluzione  $\leftarrow$  CONCATENA(RICERCA-ANGELICA(problema, gerarchia, azione), soluzione)
    sf  $\leftarrow$  si
  return soluzione

```

Figura 11.11 Un algoritmo di pianificazione gerarchica che utilizza una semantica angelica per identificare piani di alto livello che funzionano, evitando piani di alto livello che non funzionano. Il predicato PROGRESSO controlla che non ci blocchiamo in una regressione infinita di raffinamenti. Inizialmente chiamate RICERCA-ANGELICA con [Agisci] come *pianoiniziale*.

giungibili ottimistici e pessimistici con l'obiettivo (i dettagli su come calcolare gli insiemi raggiungibili di un piano, date descrizioni approssimate di ogni passo, sono trattati nell'Esercizio 11.HLAP).

Quando si trova un piano astratto che funziona, l'algoritmo *scomponere* il problema originale in sottoproblemi, uno per ogni passo del piano. Lo stato iniziale e l'obiettivo per ogni sottoproblema si ottengono mediante la regressione di uno stato obiettivo garantito come raggiungibile attraverso gli schemi di azione per ogni passo del piano (cfr. il Paragrafo 11.2.2 per spiegazioni sul funzionamento della regressione). La Figura 11.9(b) illustra l'idea di base: lo stato cerchiato a destra è lo stato obiettivo garantito come raggiungibile, quello cerchiato a sinistra è l'obiettivo intermedio ottenuto mediante regressione dell'obiettivo attraverso l'azione finale.

La capacità di affidarsi a piani di alto livello o rifiutarli può fornire a RICERCA-ANGELICA un notevole vantaggio computazionale rispetto a RICERCA-GERARCHICA, che a sua volta potrebbe avere un ampio vantaggio sul vecchio RICERCA-IN-AMPIEZZA. Consideriamo per esempio la pulizia di un grande mondo dell'aspirapolvere costituito da una disposizione di

stanze connesse da corridoi stretti, in cui ogni stanza è un rettangolo di $w \times h$ quadrati. Ha senso avere una HLA per *Naviga* (Figura 11.7) e una per *PulisciInteraStanza* (la pulizia della stanza si potrebbe implementare con l'applicazione ripetuta di un'altra HLA per pulire ogni riga). Poiché ci sono cinque azioni primitive in questo dominio, il costo per RICERCA-IN-AMPIEZZA cresce come 5^d , dove d è la lunghezza della soluzione più breve (circa due volte il numero totale dei riquadri); l'algoritmo non è in grado di gestire nemmeno due stanze 3×3 . RICERCA-GERARCHICA è più efficiente, ma soffre anch'esso di crescita esponenziale, perché tenta tutti i modi per pulire consistenti con la gerarchia. RICERCA-ANGELICA cresce più o meno linearmente nel numero dei riquadri, poiché si affida a una buona sequenza di alto livello di passi di pulizia delle stanze e di navigazione e scarta tutte le altre opzioni.

Effettuare la pulizia di un insieme di stanze pulendole una per volta non è poi così difficile: per gli esseri umani è facile proprio per la struttura gerarchica dell'attività. Quando consideriamo la difficoltà che gli esseri umani trovano nel risolvere piccoli rompicapi come quello a 8 tasselli, sembra che la capacità dell'uomo di risolvere problemi complessi derivi non dalla considerazione di elementi combinatori, ma anzi dalla sua capacità di astrarre e scomporre i problemi per eliminare elementi combinatori.

L'approccio angelico può essere esteso per trovare soluzioni di costo minimo generalizzando il concetto di insieme raggiungibile. Invece di descrivere uno stato come raggiungibile o meno, per ogni stato si avrà un costo per il cammino più efficiente che porta a raggiungerlo (il costo è infinito per stati irraggiungibili). Le descrizioni ottimistiche e pessimistiche limitano questi costi. In questo modo, la ricerca angelica può trovare piani astratti che si possono dimostrare ottimi senza dover considerare le loro implementazioni. Lo stesso approccio può essere utilizzato per ottenere algoritmi di **ricerca in avanti gerarchica** efficaci per la ricerca online, nello stile di LRTA* (Figura 4.24 del Capitolo 4).

ricerca in avanti gerarchica

In un certo senso, tali algoritmi riflettono aspetti della deliberazione umana in attività quali la pianificazione di una vacanza alle Hawaii: la considerazione di alternative è svolta inizialmente a un livello astratto su lunghe scale temporali; alcune parti del piano, per esempio come spendere pigramente due giorni a Molokai, sono lasciate astratte fino al momento dell'esecuzione, mentre altre parti sono pianificate in dettaglio, come i voli da prendere e le camere da prenotare. Senza questi raffinamenti, non vi è garanzia che il piano sia praticabile.

11.5 Pianificazione e azione in ambienti non deterministici

In questo paragrafo estendiamo la pianificazione per gestire ambienti parzialmente osservabili, non deterministici e sconosciuti. I concetti di base sono simili a quelli del Capitolo 4, ma vi sono differenze che nascono dall'uso di rappresentazioni fattorizzate anziché di rappresentazioni atomiche. Questo influisce sul modo in cui rappresentiamo la capacità dell'agente riguardo azioni e osservazioni e anche sulla rappresentazione degli **stati-credenza** – gli insiemi dei possibili stati fisici in cui l'agente può trovarsi – per ambienti parzialmente osservabili. Possiamo anche trarre vantaggio da molti dei metodi indipendenti dal dominio descritti nel Paragrafo 11.3 per calcolare euristiche per la ricerca.

Tratteremo la **pianificazione senza sensori** (nota anche come **pianificazione conformata**) per ambienti senza osservazioni; la **pianificazione condizionale** (o di contingenza) per ambienti parzialmente osservabili e non deterministici; la **pianificazione online** e la **riplanificazione** per ambienti sconosciuti. Tutto ciò ci consentirà di affrontare problemi importanti del mondo reale.

Consideriamo questo problema: date una sedia e un tavolo, l'obiettivo è di farli corrispondere: devono avere lo stesso colore. Nello stato iniziale abbiamo due barattoli di vernice, ma i colori della vernice e dei mobili sono sconosciuti.

Soltanto il tavolo inizialmente si trova nel campo visivo dell'agente:

$$\begin{aligned} & \text{Init(Oggetto(Tavolo) \wedge Oggetto(Sedia) \wedge Barattolo(C_1) \wedge Barattolo(C_2) \wedge InVista(Tavolo))} \\ & \quad \text{Obiettivo(Colore(Sedia, c) \wedge Colore(Tavolo, c))}. \end{aligned}$$

Ci sono due azioni: rimuovere il coperchio da un barattolo di vernice e verniciare un oggetto utilizzando la vernice di un barattolo aperto.

$$\begin{aligned} & \text{Azione(RimuoviCoperchio(barattolo),} \\ & \quad \text{PRECOND: Barattolo(barattolo)} \\ & \quad \text{EFFETTO: Aperto(barattolo))} \\ & \text{Azione(Vernicia(x, barattolo),} \\ & \quad \text{PRECOND: Oggetto(x) \wedge Barattolo(barattolo) \wedge Colore(barattolo, c) \wedge} \\ & \quad \quad \text{Aperto(barattolo)} \\ & \quad \text{EFFETTO: Colore(x, c))} \end{aligned}$$

Gli schemi di azione sono immediati, con una sola eccezione: precondizioni ed effetti potrebbero contenere variabili che non fanno parte della lista di variabili dell'azione. Ovvero, *Vernicia(x, barattolo)* non menziona la variabile *c*, che rappresenta il colore della vernice contenuta nel barattolo. Nel caso completamente osservabile ciò non è ammesso: dovremmo utilizzare *Vernicia(x, barattolo, c)*, ma nel caso parzialmente osservabile, non è detto che conosciamo il colore della vernice nel barattolo.

Per risolvere un problema parzialmente osservabile, l'agente dovrà ragionare sulle percezioni che otterrà mentre esegue il piano. Le percezioni saranno fornite dai sensori dell'agente quando questi sta effettivamente agendo, ma quando sta pianificando, gli servirà un modello dei suoi sensori. Nel Capitolo 4 questo modello era fornito da una funzione, *PERCEZIONE(s)*; per la pianificazione, ampliamo il linguaggio PDDL con un nuovo tipo di schema, lo **schemma di percezione**:

$$\begin{aligned} & \text{Percezione(Colore(x, c),} \\ & \quad \text{PRECOND: Oggetto(x) \wedge InVista(x))} \\ & \text{Percezione(Colore(barattolo, c),} \\ & \quad \text{PRECOND: Barattolo(barattolo) \wedge InVista(barattolo) \wedge Aperto(barattolo))}. \end{aligned}$$

schema
di percezione

Il primo schema dice che, ogni volta che un oggetto è in vista, l'agente ne percepisce il colore (questo significa che, per l'oggetto *x*, l'agente apprenderà il valore di verità di *Colore(x, c)* per tutti i *c*). Il secondo schema dice che, se un barattolo aperto è in vista, allora l'agente percepisce il colore della vernice in esso contenuta. Poiché in questo mondo non ci sono eventi esogeni, il colore di un oggetto rimarrà invariato, anche se non è percepito, finché l'agente esegue un'azione che lo cambia. Naturalmente l'agente avrà bisogno di un'azione che faccia entrare nel suo campo visivo gli oggetti (uno alla volta):

$$\begin{aligned} & \text{Azione(Osserva(x),} \\ & \quad \text{PRECOND: InVista(y) \wedge (x \neq y)} \\ & \quad \text{EFFETTO: InVista(x) \wedge \neg InVista(y))}. \end{aligned}$$

Nel caso di un ambiente completamente osservabile avremmo uno schema *Percezione* senza precondizioni per ogni fluente. Un agente senza sensori, invece, non ha alcuno schema *Percezione*. Notate che anche un agente senza sensori può risolvere il problema della verniciatura. Una soluzione è di aprire un barattolo di vernice e utilizzarlo per verniciare sedia e tavolo, **costringendoli** ad avere lo stesso colore (anche se l'agente non sa di quale colore si tratti).

Un agente con sensori che svolge una pianificazione condizionale può generare un piano migliore: prima osserva tavolo e sedia per determinarne i colori; se sono già uguali, il piano è fatto, altrimenti osserva i barattoli di vernice: se la vernice di un barattolo ha lo stesso co-

lore di uno dei due mobili, applica tale vernice anche all’altro, altrimenti vernicia entrambi i mobili con un colore qualsiasi.

Infine, un agente di pianificazione online potrebbe generare prima un piano condizionale con meno ramificazioni – eventualmente ignorando la possibilità che nessun barattolo contenga vernice dello stesso colore della sedia o del tavolo – e poi affrontare con la ripianificazione i problemi nel momento in cui si presentano. Potrebbe anche gestire gli errori dei propri schemi di azione. Mentre un pianificatore condizionale si limita ad assumere che gli effetti di un’azione abbiano sempre successo, ovvero che verniciare la sedia completa il compito, un agente di ripianificazione controllerebbe il risultato e creerebbe un piano aggiuntivo per rimediare a un fallimento inatteso, come un’area non verniciata o il colore originale che risulta visibile in trasparenza.

Nel mondo reale, gli agenti utilizzano più approcci combinati. I produttori di automobili vendono ruote di scorta e air bag, che sono incarnazioni fisiche di rami di un piano condizionale progettato per gestire forature o incidenti. D’altra parte, la maggior parte dei guidatori non considera mai queste possibilità: quando si presenta un problema, risponde come un agente di ripianificazione. In generale, gli agenti pianificano soltanto le contingenze che hanno conseguenze importanti e una probabilità non trascurabile di verificarsi. Perciò, un guidatore che pensa a un viaggio attraverso il deserto del Sahara deve creare piani condizionali esplicativi per tenere conto dei guasti, mentre per andare al supermercato non serve tutta quella pianificazione. Nel seguito esamineremo in maggiore dettaglio ciascuno dei tre approcci citati.

11.5.1 Pianificazione senza sensori

Il Paragrafo 4.4.1 del Capitolo 4 ha introdotto l’idea di base di cercare nello spazio degli stati-credenza una soluzione per problemi senza sensori. La conversione di un problema di pianificazione senza sensori in un problema di pianificazione con stati-credenza si effettua più o meno nello stesso modo descritto nel Paragrafo 4.4.1; le principali differenze sono che il modello di transizione fisico sottostante è rappresentato da una collezione di schemi di azione e che lo stato-credenza può essere rappresentato da una formula logica anziché da un insieme di stati esplicitamente enumerati. Assumiamo che il problema di pianificazione sottostante sia deterministico.

Lo stato-credenza iniziale per il problema di verniciatura senza sensori può ignorare i fluenti *InVista* perché l’agente non ha sensori. Inoltre, diamo per scontati i fatti invariabili *Oggetto(Tavolo) \wedge Oggetto(Sedia) \wedge Barattolo(C₁) \wedge Barattolo(C₂)* perché valgono in ogni stato-credenza. L’agente non conosce i colori dei barattoli e degli oggetti, e non sa se i barattoli sono aperti o chiusi, ma sa che oggetti e barattoli sono colorati: $\forall x \exists c Colore(x, c)$. Dopo la skolemizzazione (Paragrafo 9.5.1) otteniamo lo stato-credenza iniziale:

$$b_0 = Colore(x, C(x)) .$$

Nella pianificazione classica, dove si fa l’**ipotesi del mondo chiuso**, assumeremmo che ogni fluente non menzionato in uno stato sia falso, ma per la pianificazione senza sensori (e per quella parzialmente osservabile) dobbiamo passare a un’**ipotesi del mondo aperto**, in cui gli stati contengono fluenti positivi e negativi, e se un fluente non appare, il suo valore è sconosciuto. Di conseguenza, lo stato-credenza corrisponde esattamente all’insieme dei mondi possibili che soddisfano la formula. Dato questo stato-credenza iniziale, la seguente sequenza di azioni è una soluzione:

$$[RimuoviCoperchio(Barattolo_1), Vernicia(Sedia, Barattolo_1), Vernicia(Tavolo, Barattolo_1)] .$$

Ora vediamo come far progredire lo stato-credenza attraverso la sequenza di azioni per mostrare che lo stato-credenza finale soddisfa l’obiettivo.

Per prima cosa, notiamo che in un dato stato-credenza b , l'agente può considerare qualsiasi azione le cui precondizioni siano soddisfatte da b (le altre azioni non possono essere utilizzate perché il modello di transizione non definisce gli effetti di azioni le cui precondizioni potrebbero essere insoddisfatte). Secondo l'Equazione (4.4) del Capitolo 4, la formula generale per aggiornare lo stato-credenza b data un'azione applicabile a in un mondo deterministico è:

$$b' = \text{RISULTATO}(b, a) = \{s' : s' = \text{RISULTATO}_P(s, a) \text{ e } s \in b\}$$

dove RISULTATO_P definisce il modello di transizione fisico. Per il momento ipotizziamo che lo stato-credenza iniziale sia sempre una congiunzione di letterali, cioè una formula 1-CNF. Per costruire il nuovo stato-credenza b' , dobbiamo considerare che cosa accade a ciascun letterale ℓ in ogni stato fisico s in b quando si applica un'azione a . Per letterali il cui valore di verità è già noto in b , il valore di verità in b' si calcola a partire dal valore corrente e dalle liste di aggiunte e di eliminazioni dell'azione (per esempio, se ℓ è nella lista di eliminazioni dell'azione, allora $\neg\ell$ è aggiunto a b'). E che cosa possiamo dire di un letterale il cui valore di verità è sconosciuto in b ? Ci sono tre casi:

1. Se l'azione aggiunge ℓ , allora ℓ sarà vero in b' indipendentemente dal suo valore iniziale.
2. Se l'azione elimina ℓ , allora ℓ sarà falso in b' indipendentemente dal suo valore iniziale.
3. Se l'azione non interessa ℓ , allora ℓ manterrà il suo valore iniziale (che è sconosciuto) e non apparirà in b' .

Vediamo quindi che il calcolo di b' è quasi identico al caso osservabile, specificato dall'Equazione (11.1):

$$b' = \text{RISULTATO}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a).$$

Non possiamo utilizzare la semantica degli insiemi perché (1) dobbiamo assicurarsi che b' non contenga sia ℓ che $\neg\ell$ e (2) gli atomi potrebbero contenere variabili non legate. Anche qui, comunque, $\text{RISULTATO}(b, a)$ è calcolato partendo con b , impostando a falso qualsiasi atomo che appare in $\text{DEL}(a)$ e impostando a vero qualsiasi atomo che appare in $\text{ADD}(a)$. Per esempio, se applichiamo *RimuoviCoperchio(Barattolo₁)* allo stato-credenza iniziale b_0 otteniamo:

$$b_1 = \text{Colore}(x, C(x)) \wedge \text{Aperto}(\text{Barattolo}_1).$$

Quando applichiamo l'azione *Vernicia(Sedia, Barattolo₁)*, la precondizione *Colore(Barattolo₁, c)* è soddisfatta dal letterale noto *Colore(x, C(x))* con legame $\{x/ \text{Barattolo}_1, c/C(\text{Barattolo}_1)\}$ e il nuovo stato-credenza è:

$$b_2 = \text{Colore}(x, C(x)) \wedge \text{Aperto}(\text{Barattolo}_1) \wedge \text{Colore}(\text{Sedia}, C(\text{Barattolo}_1)).$$

Infine applichiamo l'azione *Vernicia(Tavolo, Barattolo₁)* per ottenere:

$$\begin{aligned} b_3 = & \text{Colore}(x, C(x)) \wedge \text{Aperto}(\text{Barattolo}_1) \wedge \text{Colore}(\text{Sedia}, C(\text{Barattolo}_1)) \\ & \wedge \text{Colore}(\text{Tavolo}, C(\text{Barattolo}_1)). \end{aligned}$$

Lo stato-credenza finale soddisfa l'obiettivo, *Colore(Tavolo, c) \wedge Colore(Sedia, c)*, con la variabile c legata a $C(\text{Barattolo}_1)$.

La precedente analisi della regola di aggiornamento ha mostrato un fatto molto importante: la famiglia di stati-credenza definiti come congiunzioni di letterali è chiusa sotto gli aggiornamenti definiti da schemi di azione PDDL. Ovvero, se lo stato-credenza inizia come congiunzione di letterali, allora ogni aggiornamento porterà a una congiunzione di letterali. Ciò significa che, in un mondo con n fluenti, ogni stato-credenza può essere rappresentato da una congiunzione di dimensione $O(n)$. Si tratta di un risultato molto confortante, considerando che il mondo contiene 2^n stati: dice che possiamo rappresentare in modo compatto tutti i sottoinsiemi di quei 2^n stati che ci serviranno. Inoltre, anche il processo di verificare



la presenza di stati-credenza che sono sottoinsiemi o sovrainsiemi di stati-credenza visitati precedentemente è facile, almeno nel caso proposizionale.

Il difetto che rovina questo bel quadro è che funziona solo per schemi di azione che hanno gli stessi effetti per tutti gli stati in cui le loro precondizioni sono soddisfatte. È questa proprietà che consente di preservare la rappresentazione dello stato credenza 1-CNF. Non appena l'effetto può dipendere dallo stato, si introducono delle dipendenze tra fluenti e la proprietà 1-CNF si perde. Consideriamo per esempio il semplice mondo dell'aspirapolvere definito nel Paragrafo 3.2.1. Indichiamo con InS e InD la posizione del robot e con $PulitoS$ e $PulitoD$ lo stato dei riquadri. Secondo la definizione del problema, l'azione *Aspira* non ha precondizione: può sempre essere eseguita. La difficoltà sta nel fatto che il suo effetto dipende dalla posizione del robot: quando questo è InS , il risultato è $PulitoS$, ma quando è InD , il risultato è $PulitoD$. Per tali azioni, i nostri schemi di azione necessitano di qualcosa di nuovo: un **effetto condizionale**. La sintassi è “**when** condizione : effetto”, dove *condizione* è una formula logica da confrontare con lo stato corrente ed *effetto* è una formula che descrive lo stato risultante. Per il mondo dell'aspirapolvere abbiamo:

Azione(Aspira),

EFFETTO: **when** InS : $PulitoS \wedge$ **when** InD : $PulitoD$.

Quando la si applica allo stato-credenza iniziale *True*, lo stato-credenza risultante è $(InS \wedge PulitoS) \vee (InD \wedge PulitoD)$, che non è più in 1-CNF (questa transizione è visibile nella Figura 4.14 del Capitolo 4). In generale, gli effetti condizionali possono indurre dipendenze arbitrarie tra i fluenti in uno stato-credenza, portando a stati-credenza di dimensione esponenziale nel caso peggiore.

È importante capire la differenza tra precondizioni ed effetti condizionali. Tutti gli effetti condizionali le cui condizioni sono soddisfatte sono applicati per generare lo stato risultante; se nessuna è soddisfatta, allora lo stato risultante è invariato. D'altra parte, se una precondizione non è soddisfatta, l'azione è inapplicabile e lo stato risultante è indefinito. Dal punto di vista della pianificazione senza sensori, è meglio avere effetti condizionali che un'azione inapplicabile. Per esempio, potremmo suddividere *Aspira* in due azioni con effetti non condizionali, come segue:

Azione(AspiraS),

PRECOND: InS ; EFFETTO: $PulitoS$)

Azione(AspiraD),

PRECOND: InD ; EFFETTO: $PulitoD$).

Oraabbiamo soltanto schemi non condizionali, perciò gli stati-credenza rimangono tutti in 1-CNF; sfortunatamente non possiamo determinare l'applicabilità di *AspiraS* e *AspiraD* nello stato-credenza iniziale.

Sembra dunque inevitabile che problemi non banali coinvolgano stati-credenza contorti, come quelli incontrati quando abbiamo considerato il problema della stima dello stato per il mondo del wumpus (Figura 7.21 del Capitolo 7). La soluzione suggerita in quel caso era di utilizzare un'**approssimazione conservativa** dello stato-credenza esatto; per esempio, lo stato-credenza può rimanere in 1-CNF se contiene tutti i letterali i cui valori di verità possono essere determinati e considera tutti gli altri letterali come sconosciuti. Questo approccio appare buono perché non genera mai un piano errato, ma è incompleto perché potrebbe non essere in grado di trovare soluzioni a problemi che comportano necessariamente interazioni tra letterali. Per fornire un esempio banale, se l'obiettivo è che il robot si trovi su un riquadro pulito, allora [*Aspira*] è una soluzione, ma un agente senza sensori che insista su stati-credenza 1-CNF non la troverà.

effetto condizionale

Forse una soluzione migliore è cercare sequenze di azioni che mantengano lo stato-credenza il più semplice possibile. Nel mondo dell'aspirapolvere senza sensori, la sequenza di azioni *[Destra, Aspira, Sinistra, Aspira]* genera la seguente sequenza di stati-credenza:

$$\begin{aligned} b_0 &= \text{True} \\ b_1 &= \text{InD} \\ b_2 &= \text{InD} \wedge \text{PulitoD} \\ b_3 &= \text{InS} \wedge \text{PulitoD} \\ b_4 &= \text{InS} \wedge \text{PulitoD} \wedge \text{PulitoS} \end{aligned}$$

Questo significa che l'agente può risolvere il problema mantenendo uno stato-credenza 1-CNF, anche se alcune sequenze (quelle che iniziano con *Aspira*) non rimangono in 1-CNF. Possiamo trarre una lezione generale per noi umani: eseguiamo sempre piccole azioni (controllare l'ora, frugare nelle tasche per assicurarsi di avere le chiavi della macchina, leggere i segnali stradali mentre si guida attraverso una città) per eliminare l'incertezza e mantenere gestibile il nostro stato-credenza.

C'è un altro approccio, piuttosto diverso, al problema degli stati-credenza troppo contorti per poter essere gestiti: non preoccuparsi di calcolarli. Supponiamo che lo stato-credenza iniziale sia b_0 e di voler conoscere lo stato-credenza risultante dalla sequenza di azioni $[a_1, \dots, a_m]$. Invece di calcolarlo esplicitamente, rappresentiamolo come " b_0 e poi $[a_1, \dots, a_m]$ ". Si tratta di una rappresentazione non precisa ma non ambigua dello stato-credenza, e anche piuttosto concisa: $O(n + m)$ dove n è la dimensione dello stato-credenza iniziale (che si suppone in 1-CNF) e m è la lunghezza massima di una sequenza di azioni. Quale rappresentazione di uno stato-credenza, tuttavia, soffre di un difetto: determinare se l'obiettivo sia soddisfatto, o se un'azione sia applicabile, potrebbe richiedere molti calcoli.

Il calcolo può essere implementato come un test di conseguenza logica: se A_m rappresenta la collezione di assiomi di stato successore richiesti per definire le occorrenze delle azioni a_1, \dots, a_m – come descritto per SATPLAN nel Paragrafo 11.2.3 – e G_m asserisce che l'obiettivo è vero dopo m passi, allora il piano raggiunge l'obiettivo se $b_0 \wedge A_m \models G_m$, cioè se $b_0 \wedge A_m \wedge \neg G_m$ è insoddisfacibile. Dato un moderno risolutore SAT, potrebbe essere possibile calcolare ciò in modo molto più rapido rispetto al calcolo dell'intero stato-credenza. Per esempio, se nessuna delle azioni della sequenza ha un particolare fluente obiettivo nella propria lista di aggiunte, il risolutore lo rileverà immediatamente. È utile, inoltre, che i risultati parziali sullo stato-credenza (per esempio, fluenti di cui si sa se sono veri o falsi) siano memorizzati in una cache per semplificare i calcoli successivi.

L'ultimo elemento per la pianificazione senza sensori è una funzione euristica che guida la ricerca. Il significato della funzione euristica è identico a quello per la pianificazione classica: una stima (forse ammissibile) del costo per raggiungere l'obiettivo dallo stato-credenza dato. Con gli stati-credenza abbiamo un fatto in più: risolvere un sottoinsieme di uno stato-credenza è necessariamente più facile che risolvere lo stato-credenza originale:

$$\text{if } b_1 \subseteq b_2 \text{ then } h^*(b_1) \leq h^*(b_2).$$

Quindi, qualsiasi euristica ammissibile calcolata per un sottoinsieme è ammissibile per lo stato-credenza. I candidati più ovvi sono i sottoinsiemi singoletti, ovvero i singoli stati fisici. Possiamo prendere qualsiasi collezione casuale di stati s_1, \dots, s_N contenuti nello stato-credenza b , applicare qualsiasi euristica ammissibile h e considerare:

$$H(b) = \max\{h(s_1), \dots, h(s_N)\}$$

come stima euristica per risolvere b . Possiamo anche utilizzare euristiche inammissibili, come quella che ignora le liste di eliminazioni (Paragrafo 11.3), che sembra funzionare piuttosto bene nella pratica.

11.5.2 Pianificazione condizionale

Nel Capitolo 4 abbiamo visto che la pianificazione condizionale (la generazione di piani con ramificazioni condizionali basate su percezioni) è appropriata per ambienti parzialmente osservabili, non deterministici o entrambi. Per il problema di verniciatura parzialmente osservabile con gli assiomi di percezione forniti in precedenza, una possibile soluzione condizionale è la seguente:

```
[Osserva(Tavolo), Osserva(Sedia),
  if Colore(Tavolo, c) ∧ Colore(Sedia, c) then NoOp
  else [RimuoviCoperchio(Barattolo1), Osserva(Barattolo1), RimuoviCoperchio
        (Barattolo2), Osserva(Barattolo2) ,
        if Colore(Tavolo, c) ∧ Colore(barattolo, c) then Vernicia(Sedia, barattolo)
        else if Colore(Sedia, c) ∧ Colore(barattolo, c) then Vernicia(Tavolo,
                           barattolo)
        else [Vernicia(Sedia, Barattolo1), Vernicia(Tavolo, Barattolo1)]]].
```

Le variabili di questo piano devono essere considerate quantificate esistenzialmente; la seconda riga dice che, se esiste un colore c che è il colore del tavolo e della sedia, allora l'agente non deve fare nulla per raggiungere l'obiettivo. Quando esegue questo piano, un agente di pianificazione condizionale può mantenere il proprio stato-credenza come formula logica e valutare ogni condizione di ramificazione determinando se dallo stato-credenza segue logicamente la formula della condizione o la sua negazione (spetta all'algoritmo di pianificazione condizionale assicurarsi che l'agente non entrerà mai in uno stato-credenza in cui il valore di verità della formula della condizione è sconosciuto). Notate che, con condizioni del primo ordine, la formula potrebbe essere soddisfatta in più modi; per esempio, la condizione $\text{Colore}(\text{Tavolo}, c) \wedge \text{Colore}(\text{barattolo}, c)$ potrebbe essere soddisfatta da $\{\text{barattolo}/\text{Barattolo}_1\}$ e da $\{\text{barattolo}/\text{Barattolo}_2\}$ se entrambi i barattoli contengono vernice dello stesso colore del tavolo. In tal caso, l'agente può scegliere qualsiasi sostituzione che soddisfi la condizione e applicarla al resto del piano.

Come si è mostrato nel Paragrafo 4.4.2, per calcolare il nuovo stato-credenza \hat{b} dopo un'azione a e la successiva percezione si procede in due passi. Il primo passo calcola lo stato-credenza dopo l'azione, come per l'agente senza sensori:

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

dove, come prima, abbiamo assunto uno stato-credenza rappresentato come congiunzione di letterali. Il secondo passo è più complesso; supponiamo che siano ricevuti i letterali di percezione p_1, \dots, p_k . Si potrebbe pensare che basti semplicemente aggiungere questi letterali nello stato-credenza; ma in pratica possiamo anche inferire che le precondizioni per le percezioni sono soddisfatte. Ora, se una percezione p ha esattamente uno schema di percezione, $\text{Percezione}(p, \text{PRECOND}: c)$, dove c è una congiunzione di letterali, allora questi letterali possono essere inseriti nello stato-credenza insieme a p . D'altra parte, se p ha più di uno schema di percezione le cui precondizioni potrebbero valere nello stato-credenza predetto \hat{b} , allora dobbiamo aggiungere la disgiunzione delle precondizioni. Ovviamente questo porta lo stato-credenza al di fuori della forma 1-CNF e fa nascere le stesse complicazioni degli effetti condizionali, con più o meno le stesse classi di soluzioni.

Dato un meccanismo per calcolare stati-credenza esatti o approssimati, possiamo generare piani condizionali con un'estensione della ricerca in avanti AND-OR sugli stati-credenza utilizzata nel Paragrafo 4.4. Le azioni con effetti non deterministici – definite utilizzando semplicemente una disgiunzione nell'EFFETTO dello schema di azione – possono essere inserite con lievi modifiche nel calcolo dell'aggiornamento dello stato-credenza, senza modi-

ficare l'algoritmo di ricerca.³ Per la funzione euristica, molti dei metodi suggeriti per la pianificazione senza sensori sono applicabili anche nel caso non deterministico parzialmente osservabile.

11.5.3 Pianificazione online

Immaginate di osservare un robot che esegue saldature in un impianto industriale per la produzione di automobili. I movimenti veloci e precisi del robot sono ripetuti via via che ogni auto passa nella linea di montaggio. Benché sia tecnicamente impressionante, il robot probabilmente non sembra affatto intelligente perché il movimento è una sequenza fissa, preprogrammata; il robot ovviamente non “sa che cosa sta facendo”, in alcun senso. Ora supponete che una portiera montata male si stacchi dall’auto e cada proprio mentre il robot sta per effettuare una saldatura. Il robot sostituisce rapidamente il proprio strumento di saldatura con uno strumento per afferrare oggetti, raccoglie la porta, controlla la presenza di graffi, la riapplica all’auto, invia un’email al supervisore, riapplica lo strumento di saldatura e riprende il proprio lavoro. Improvvisamente il comportamento del robot sembra *intenzionale*, anziché puramente meccanico; ipotizziamo che tale comportamento nasca non da un grande piano condizionale precalcolato, ma da un processo di ripianificazione online, il che significa che il robot *deve sapere* che cosa sta cercando di fare.

La ripianificazione presuppone una forma di **monitoraggio dell’esecuzione** per determinare la necessità di un nuovo piano. Tale necessità si presenta quando un agente di pianificazione condizionale si stanca di pianificare per ogni piccola contingenza, come la domanda se il cielo potrebbe cadere sulla sua testa.⁴ Per esempio, alcuni rami di un piano condizionale parzialmente costruito possono indicare semplicemente *Ripianifica*; se si raggiunge un tale ramo durante l’esecuzione, l’agente rientra in modalità di pianificazione. Come abbiamo detto in precedenza, la decisione relativa a quanta parte del problema risolvere in anticipo e quanta parte lasciare alla ripianificazione comporta dei compromessi tra eventi possibili con costi diversi e probabilità che si verifichino. Nessuno vuole che la propria auto si guasti nel mezzo del deserto del Sahara per chiedersi, soltanto allora, se c’è abbastanza acqua.

Potrebbe essere necessaria la ripianificazione se l’agente ha un modello del mondo errato. Il modello per un’azione potrebbe presentare una **precondizione mancante**: per esempio, l’agente potrebbe non sapere che per rimuovere il coperchio di un barattolo di vernice spesso serve un cacciavite. Il modello potrebbe presentare un **effetto mancante**: per esempio, la verniciatura di un oggetto potrebbe provocare la caduta di vernice sul pavimento. Oppure il modello potrebbe presentare un **fluente mancante**, cioè assente dalla rappresentazione; per esempio, il modello fornito precedentemente non ha la nozione della quantità di vernice contenuta in un barattolo, o di come le sue azioni influiscono su tale quantità, o della necessità che tale quantità non sia nulla. Il modello potrebbe anche non considerare **eventi esogeni**, per esempio qualcuno che rovescia il barattolo. Gli eventi esogeni possono anche comprendere variazioni dell’obiettivo, quale l’aggiunta del requisito che il tavolo e la sedia non siano verniciati di nero. Senza la capacità di monitorare e ripianificare, il comportamento di

**monitoraggio
dell’esecuzione**

**precondizione
mancante**
effetto mancante

fluente mancante

evento esogeno

³ Se sono richieste soluzioni cicliche per un problema non deterministico, la ricerca AND-OR deve essere generalizzata in una versione ciclica quale LAO* (Hansen e Zilberstein, 2001).

⁴ Nel 1954 la signora Hodges dell’Alabama fu colpita da un meteorite che sfondò il tetto della sua casa. Nel 1992 un pezzo del meteorite Mbale colpì un ragazzino sulla testa; fortunatamente la sua discesa fu rallentata da foglie di banano (Jenniskens et al., 1994). Nel 2009, un ragazzo tedesco affermò di essere stato colpito sulla mano da un meteorite grande quanto un pisello. Nessuno di questi incidenti ha provocato ferite gravi, il che suggerisce che la necessità di pianificare rispetto a tali contingenze è talvolta sovrastimata.

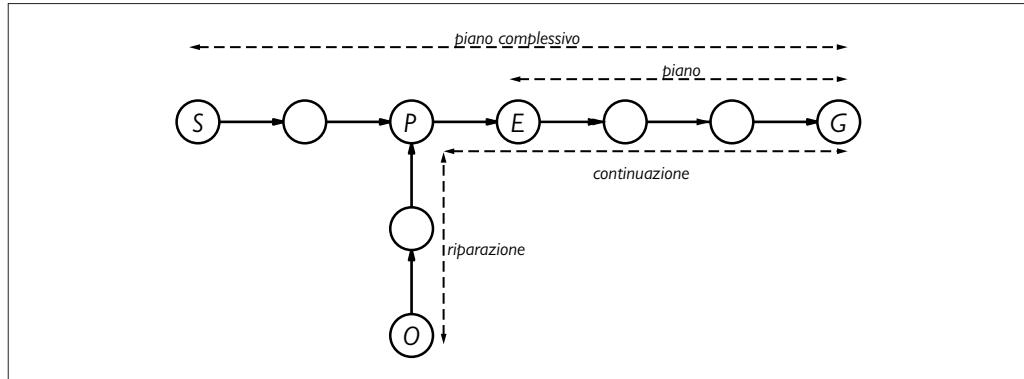


Figura 11.12 Inizialmente la sequenza “piano complessivo” dovrebbe portare l’agente da S a G . L’agente esegue il piano finché si aspetta di trovarsi nello stato E , ma osserva che in realtà si trova in O . A questo punto ripianifica costruendo la *riparazione* più la *continuazione* per raggiungere G di lunghezza complessiva minima.

un agente sarà con tutta probabilità fragile, se si basa sull’assoluta correttezza del proprio modello.

Un agente online può scegliere tra (almeno) tre approcci diversi per monitorare l’ambiente durante l’esecuzione:

- monitoraggio dell’azione
- monitoraggio del piano
- monitoraggio dell’obiettivo

- **Monitoraggio dell’azione:** prima di eseguire un’azione, l’agente verifica che tutte le precondizioni valgano.
- **Monitoraggio del piano:** prima di eseguire un’azione, l’agente verifica che la parte rimanente del piano porti ancora al successo.
- **Monitoraggio dell’obiettivo:** prima di eseguire un’azione, l’agente controlla se esiste un insieme di obiettivi migliore che potrebbe tentare di raggiungere.

La Figura 11.12 raffigura schematicamente il monitoraggio di un’azione. L’agente mantiene traccia del piano originale, *piano complessivo*, e della parte del piano che non è ancora stata eseguita, denotata da *piano*. Dopo aver eseguito i primi passi del piano, l’agente si aspetta di trovarsi in E , tuttavia osserva che in realtà si trova nello stato O . Deve quindi correggere il piano trovando un punto P sul piano originale a cui può tornare (può anche darsi che P sia lo stato obiettivo G). L’agente cerca di minimizzare il costo totale del piano: la parte della riparazione (da O a P) più la continuazione (da P a G).

Ora torniamo al problema di esempio in cui si vuole ottenere una sedia e un tavolo dello stesso colore. Supponiamo che l’agente elabori il seguente piano:

```
[Osserva(Tavolo), Osserva(Sedia),
  if Colore(Tavolo, c) ∧ Colore(Sedia, c) then NoOp
  else [RimuoviCoperchio(Barattolo1), Osserva(Barattolo1),
    if Colore(Tavolo, c) ∧ Colore(Barattolo1, c) then Vernicia(Sedia, Barattolo1)
    else RIPIANIFICA]] .
```

Ora l’agente è pronto a eseguire il piano. L’agente osserva che il tavolo e il barattolo di vernice sono bianchi e la sedia è nera. Esegue quindi *Vernicia(Sedia, Barattolo₁)*. A questo punto un pianificatore classico dichiarerebbe vittoria: il piano è stato eseguito. Ma un agente online con monitoraggio dell’esecuzione deve verificare che l’azione abbia avuto successo.

Supponiamo che l’agente percepisca che la sedia è screziata di grigio perché la vernice nera si vede in trasparenza. Allora deve determinare un punto da scegliere come obiettivo parziale e trovare una sequenza di azioni di riparazione per raggiungerlo. L’agente nota che lo stato corrente è identico alla precondizione dell’azione *Vernicia(Sedia, Barattolo₁)*, così

sceglie una sequenza vuota come *riparazione* e imposta come *piano* la stessa sequenza [*Vernicia*] appena tentata. A questo punto il monitoraggio dell'esecuzione riprende e l'azione *Vernicia* è eseguita nuovamente. Questo comportamento si ripeterà ciclicamente finché la sedia non sarà percepita come completamente verniciata. Notate però che il ciclo è originato dal processo pianificazione-esecuzione-riplanificazione, e dalla presenza di un ciclo esplicito all'interno di un piano. Inoltre, il piano originale non deve necessariamente coprire ogni contingenza. Se l'agente raggiunge il passo contrassegnato con **RIPIANIFICA**, può generare un nuovo piano (che potrebbe coinvolgere *Barattolo*₂).

Il monitoraggio dell'azione è un metodo semplice di monitoraggio dell'esecuzione, ma può talvolta portare a un comportamento non intelligente. Per esempio, supponiamo che non vi sia vernice nera o bianca, e che l'agente costruisca un piano per risolvere il problema verniciando di rosso la sedia e il tavolo. Supponiamo che vi sia vernice sufficiente soltanto per la sedia. Con il monitoraggio dell'azione, l'agente proseguirebbe e vernicerebbe la sedia di rosso, poi noterebbe di aver esaurito la vernice e non potrebbe verniciare il tavolo, e a quel punto ripianificherebbe una riparazione, forse la verniciatura di sedia e tavolo in verde. Un agente con monitoraggio del piano può rilevare il fallimento ogni volta che lo stato corrente è tale che la parte rimanente del piano non funziona più, quindi non sprecerebbe tempo a verniciare la sedia di rosso. Il monitoraggio del piano è in grado di fare ciò controllando le precondizioni per tutta la parte rimanente del piano, ovvero per ogni passo del piano, eccetto le precondizioni soddisfatte da un altro passo nella parte rimanente. In questo modo si arresta il prima possibile l'esecuzione di un piano senza sbocco, anziché continuare fino al verificarsi dell'inevitabile fallimento.⁵

Il monitoraggio del piano consente anche la **serendipità**, o successo casuale. Se interviene qualcuno che vernicia il tavolo di rosso nello stesso tempo in cui l'agente vernicia la sedia di rosso, le precondizioni del piano finale sono soddisfatte (l'obiettivo è stato raggiunto) e l'agente può terminare prima.

È facile modificare un algoritmo di pianificazione in modo che ogni azione del piano sia annotata con le proprie precondizioni, così da consentire il monitoraggio dell'azione. È leggermente più difficile abilitare il monitoraggio del piano. I pianificatori con ordinamento parziale hanno il vantaggio di avere già integrate strutture che contengono le relazioni necessarie per il monitoraggio del piano. Espandere i pianificatori che operano nello spazio degli stati con le necessarie annotazioni è possibile con un attento controllo che annoti la regressione dei fluenti obiettivo attraverso il piano.

Ora che abbiamo descritto un metodo per il monitoraggio e la ripianificazione, dobbiamo chiederci se funziona. La questione è sorprendentemente complessa. Se intendiamo: "possiamo garantire che l'agente raggiungerà sempre l'obiettivo?" la risposta è no, perché l'agente potrebbe inavvertitamente arrivare in un vicolo cieco da cui non vi è riparazione possibile. Per esempio, l'agente del mondo dell'aspirapolvere potrebbe avere un modello errato di sé e non sapere che le sue batterie possono esaurirsi, nel qual caso non può correggere alcun piano. Se eliminiamo i vicoli ciechi, supponendo che da *qualsiasi* stato dell'ambiente esista un piano per raggiungere l'obiettivo, e assumiamo che l'ambiente sia realmente non deterministico, nel senso che un tale piano ha sempre *qualche* possibilità di successo in qualsiasi tentativo di esecuzione, allora l'agente alla fine raggiungerà l'obiettivo.

I problemi nascono quando un'azione che appare non deterministica in realtà non è realmente casuale, ma dipende da una precondizione che l'agente non conosce. Per esempio,

⁵ Il monitoraggio del piano significa che finalmente, giunti a questo punto, abbiamo un agente più intelligente dello scarabeo stercoreo (cfr. Paragrafo 2.2.3). Un agente con monitoraggio del piano noterebbe che la pallina di sterco non è più in suo possesso ed eseguirebbe una ripianificazione per procurarsene un'altra con cui tappare l'ingresso della tana.

talvolta un barattolo di vernice potrebbe essere vuoto, perciò verniciare con quel barattolo non avrebbe alcun effetto, e nessun nuovo tentativo potrebbe cambiare questo fatto.⁶ Una soluzione è di scegliere a caso nell'insieme dei possibili piani di riparazione, anziché provare lo stesso ogni volta. In questo caso, il piano di riparazione che indica di aprire un altro barattolo potrebbe funzionare. Un approccio migliore è quello di **apprendere** un modello migliore. Ogni fallimento di una previsione è un'opportunità di apprendimento; un agente dovrebbe essere in grado di modificare il proprio modello del mondo secondo le proprie percezioni. Da qui, il ripianificatore sarà in grado di costruire una riparazione che arrivi alla radice del problema, anziché affidarsi alla fortuna.

11.6 Tempo, scheduling e risorse

scheduling
vincoli di risorse

La pianificazione classica parla di *cosa fare* e in *quale ordine*, ma non dice nulla riguardo il tempo: *quanto dura* un'azione e *quando* si verifica. Per esempio, nel dominio dell'aeroporto potremmo produrre un piano che dica quali aerei vanno verso quale destinazione, che cosa portano, senza però specificare gli orari di partenza e di arrivo. Di questo si occupa lo **scheduling**.

Il mondo reale, inoltre, impone molti **vincoli di risorse**: una linea aerea ha un numero limitato di equipaggi, e un equipaggio che si trova su un volo non può trovarsi contemporaneamente su un altro. In questo paragrafo introduciamo tecniche per affrontare problemi di pianificazione e scheduling con vincoli di risorse.

L'approccio che adottiamo qui è del tipo “prima pianifica, poi passa allo scheduling”: dividiamo il problema complessivo in una fase di pianificazione, con alcuni vincoli di ordinamento, per soddisfare gli obiettivi del problema, e una successiva fase di *scheduling* in cui si aggiungono al piano informazioni temporali per assicurarsi che soddisfi i vincoli di risorse e le scadenze. Questo approccio è comune negli ambienti manifatturieri e logistici del mondo reale, dove la fase di pianificazione è spesso automatizzata e a volte effettuata da esperti umani.

11.6.1 Rappresentazione di vincoli temporali e di risorse

job-shop scheduling

job

durata

consumabile

riusabile

makespan

Un tipico problema di **job-shop scheduling** (o programmazione di lavori, cfr. Paragrafo 6.1.2) consiste di un insieme di **job** (lavori), ognuno dei quali ha una collezione di **azioni** con vincoli di ordinamento tra di esse. Ogni azione ha una **durata** e un insieme di vincoli sulle risorse richieste. Un vincolo specifica un tipo di risorsa (per esempio bulloni, chiavi o piloti), il numero di tali risorse richieste e se la risorsa è **consumabile** (per esempio, i bulloni non sono più disponibili all'uso) o **riusabile** (per esempio, un pilota è occupato durante un volo, ma è di nuovo disponibile quando il volo è terminato). Le azioni possono anche produrre risorse (per esempio le azioni di produzione e riapprovvigionamento).

Una soluzione di un problema di job-shop scheduling specifica i tempi di inizio per ciascuna azione e deve soddisfare tutti i vincoli di ordinamento temporale e di risorse. Come per i problemi di ricerca e di pianificazione, le soluzioni possono essere valutate secondo una funzione di costo, ma questo può risultare piuttosto complicato con costi delle risorse non lineari, costi di ritardo dipendenti dal tempo e così via. Per semplicità assumiamo che la funzione di costo sia semplicemente la durata totale del piano, che si chiama **makespan**.

La Figura 11.13 mostra un semplice esempio: un problema che riguarda l'assemblaggio di due auto. Il problema è costituito da due job, ognuno della forma *[AggiungiMotore, AggiungiRuote, Ispeziona]*. Poi l'istruzione *Risorse* dichiara che vi sono quattro tipi di risorse e fornisce il numero di risorse di ciascun tipo disponibili all'inizio: 1 sollevatore per motori,

⁶ La ripetizione futile della riparazione di un piano è precisamente il comportamento esibito dalla vespa sphex (cfr. Paragrafo 2.2.3).

```

Job({AggiungiMotore1 < AggiungiRuote1 < Ispeziona1},
    {AggiungiMotore2 < AggiungiRuote2 < Ispeziona2})

Risorse(SollevaMotori(1), SupportoRuote(1), Ispettore(2), Bulloni(500))

Azione(AggiungiMotore1, DURATA: 30
      USO: SollevaMotori(1))
Azione(AggiungiMotore2, DURATA: 60
      USO: SollevaMotori(1))
Azione(AggiungiRuote1, DURATA: 30
      CONSUMO: Bulloni(20), USO: SupportoRuote(1))
Azione(AggiungiRuote2, DURATA: 15
      CONSUMO: Bulloni(20), USO: SupportoRuote(1))
Azione(Ispezionai, DURATA: 10
      USO: Ispettore(1))

```

1 supporto ruote, 2 ispettori e 500 bulloni per le ruote. Gli schemi di azione forniscono la durata e le risorse richieste per ciascuna azione. I bulloni sono consumati quando si aggiungono le ruote all'auto, mentre le altre risorse sono “prese in prestito” all'inizio di un'azione e rilasciate al termine.

La rappresentazione delle risorse come quantità numeriche, come *Ispettore(2)*, anziché entità dotate di nome, come *Ispettore(I₁)* e *Ispettore(I₂)*, è un esempio di una tecnica chiamata **aggregazione** che consiste nel raggruppare oggetti distinti in quantità collettive quando sono tutti indistinguibili. Nel nostro problema di assemblaggio non importa *quale* ispettore esamina le macchine, per cui non c'è alcuna ragione di distinguere tra i due. L'aggregazione è fondamentale per ridurre la complessità. Considerate che cosa succede quando si propone uno schedule che ha 10 azioni *Ispeziona* concorrenti, ma sono disponibili solo 9 ispettori. Quando gli ispettori sono rappresentati come quantità il rilevamento del fallimento è istantaneo, e l'algoritmo può tornare indietro con il backtracking e provare uno schedule diverso. Se gli ispettori fossero rappresentati con oggetti singoli, l'algoritmo proverebbe inutilmente tutti i 9! modi di assegnarli alle azioni, prima di accorgersi che nessuno di essi funziona.

11.6.2 Risoluzione di problemi di scheduling

Iniziamo considerando soltanto il problema di scheduling temporale e ignorando i vincoli di risorse. Per minimizzare il *makespan* (la durata del piano), dobbiamo trovare i più precoci tempi di inizio per tutte le azioni consistenti con i vincoli di ordinamento forniti con il problema. È utile visualizzare questi vincoli di ordinamento con un grafo orientato che mette in relazione fra loro le azioni (Figura 11.14). Possiamo applicare il **metodo del cammino critico** (CPM, dall'inglese *critical path method*) a tale grafo per determinare i possibili istanti di inizio e fine di ogni azione. Un **cammino** attraverso un grafo che rappresenta un piano con ordinamento parziale è una sequenza linearmente ordinata di azioni che comincia con *Inizio* e termina con *Fine*: per esempio, nel piano con ordinamento parziale della Figura 11.14 ci sono due cammini.

Il **cammino critico** è quello con la durata totale più lunga. Si chiama “critico” perché determina la durata dell'intero piano: accorciare altri cammini non accorcia il piano globale, mentre rimandare l'inizio di una qualsiasi azione sul cammino critico lo rallenta. Le azioni poste al di fuori del cammino critico hanno una finestra temporale in cui possono essere ese-

Figura 11.13
Un problema di job-shop scheduling per l'assemblaggio di due automobili, con vincoli di risorse. La notazione $A \prec B$ significa che l'azione A deve precedere l'azione B.

aggregazione

metodo del cammino critico

cammino critico

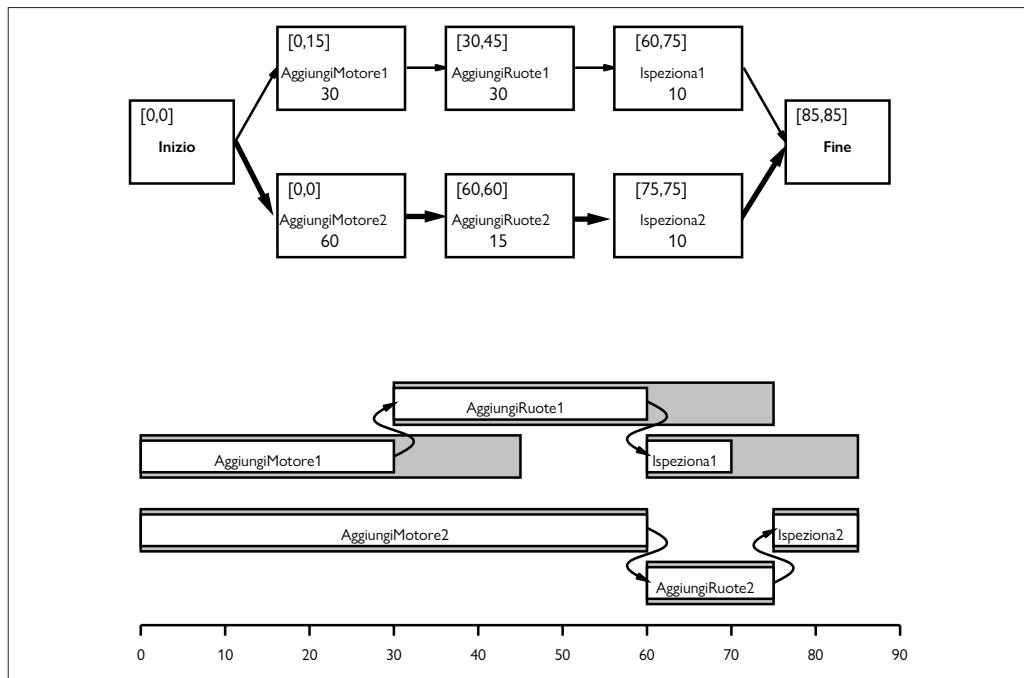


Figura 11.14 In alto: una rappresentazione dei vincoli temporali per il problema di job-shop scheduling della Figura 11.13. La durata di ogni azione è specificata nella parte inferiore di ogni rettangolo. Nella risoluzione del problema, calcoliamo il primo e l'ultimo tempo iniziale come coppia $[ES, LS]$, riportata in alto a sinistra. La differenza tra questi due numeri è il **margine** dell'azione; le azioni con margine zero si trovano sul cammino critico, evidenziato da linee più spesse. In basso: la stessa soluzione rappresentata come una timeline. I rettangoli grigi rappresentano gli intervalli temporali durante i quali un'azione può essere eseguita, a patto che siano rispettati i vincoli di ordinamento. La parte libera di un rettangolo grigio indica il margine.

margine
schedule

guite. Per specificare tale finestra si deve indicare il primo istante in cui è possibile iniziare l'azione (*ES*, da *Earliest Start*) e l'ultimo (*LS*, da *Latest Start*). La quantità $LS - ES$ è denominata **margine** (*slack*) dell'azione. Possiamo vedere nella Figura 11.14 che l'intero piano richiederà 85 minuti, che ogni azione nel job superiore ha 15 minuti di margine e che ogni azione sul cammino critico non ha margine (per definizione). Insieme, gli *ES* e *LS* di tutte le azioni costituiscono uno **schedule** del problema.

Le seguenti formule definiscono *ES* e *LS* e costituiscono un algoritmo di programmazione dinamica capace di calcolarle. *A* e *B* sono azioni e $A < B$ significa che *A* precede *B*:

$$\begin{aligned} ES(\text{Inizio}) &= 0 \\ ES(B) &= \max_{A < B} ES(A) + \text{Durata}(A) \\ LS(\text{Fine}) &= ES(\text{Fine}) \\ LS(A) &= \min_{B > A} LS(B) - \text{Durata}(A). \end{aligned}$$

L'idea è di cominciare assegnando 0 a $ES(\text{Inizio})$. Non appena abbiamo un'azione *B* tale che tutte le azioni che vengono immediatamente prima hanno dei valori di *ES* assegnati, imponiamo che $ES(B)$ sia il valore massimo tra i tempi finali più bassi di tutte le azioni immediatamente precedenti a *B*, dove il "tempo finale più basso" di un'azione è definito come il suo *ES* più la sua durata. Questo processo si ripete finché è stato assegnato un valore *ES* a ogni azione. I valori *LS* sono calcolati in modo simile, muovendosi all'indietro dall'azione *Fine*.

La complessità dell'algoritmo del cammino critico è solo $O(Nb)$, dove *N* è il numero di azioni e *b* il fattore di ramificazione massimo in ingresso o uscita da un'azione (per consta-

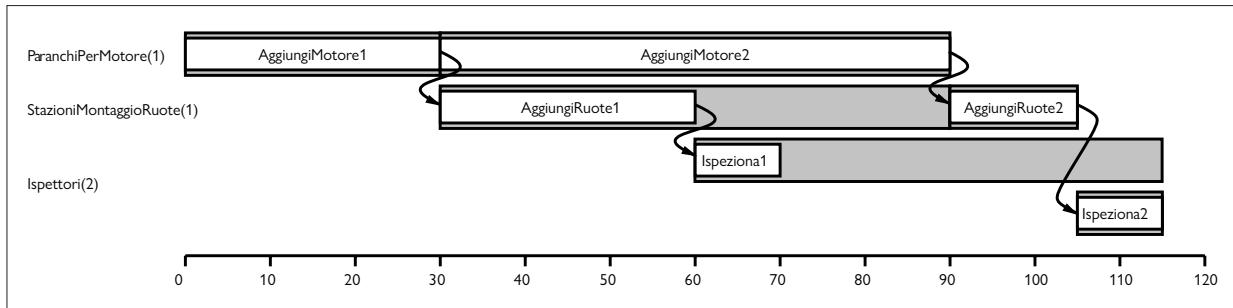


Figura 11.15 Una soluzione del problema di job-shop scheduling della Figura 11.13 che tiene conto dei vincoli di risorse. A sinistra sono elencate le tre risorse riusabili, le azioni sono allineate orizzontalmente con le risorse che usano. Ci sono due possibili schedule, a seconda di quale linea di assemblaggio usa il sollevatore di motori per prima; abbiamo mostrato la soluzione di durata minima, che richiede 115 minuti.

tarlo, considerate che il calcolo di *LS* e *ES* viene effettuato una volta per ogni azione, e ogni volta itera al più su *b* altre azioni). Di conseguenza, trovare uno schedule di minima durata, dato un ordinamento parziale delle azioni e nessun vincolo di risorse, è abbastanza facile.

In termini matematici, i problemi di cammino critico sono facili da risolvere perché sono definiti come *congiunzione* di disuguaglianze *lineari* sugli istanti iniziale e finale. Quando introduciamo vincoli di risorse, i vincoli risultanti sugli instanti iniziale e finale diventano più complicati. Per esempio, le azioni *AggiungiMotore*, che iniziano allo stesso tempo nella Figura 11.14, richiedono lo stesso *SollevaMotori* e perciò non possono sovrapporsi. Il vincolo “non possono sovrapporsi” è una *disgiunzione* di due disuguaglianze lineari, una per ogni possibile ordinamento. L’introduzione di disgiunzioni rende NP-difficili i problemi di scheduling con vincoli di risorse.

La Figura 11.15 mostra la soluzione con il minor tempo cumulativo, 115 minuti. Questo tempo è maggiore di 30 minuti rispetto agli 85 minuti richiesti dalla schedule privo di vincoli sulle risorse. Notate che non c’è alcun momento in cui entrambi gli ispettori sono necessari, per cui è possibile rimuoverne immediatamente uno e assegnarlo a una mansione più produttiva.

Vi è una lunga tradizione di lavori sullo scheduling ottimo. Un problema dimostrativo posto nel 1963 – trovare lo schedule ottimo per un problema con 10 macchine e 10 job, ognuno di 100 azioni – non fu risolto prima di 23 anni (Lawler *et al.*, 1993). Per ridurre la complessità sono stati provati molti approcci, tra cui branch-and-bound, simulated annealing, ricerca tabù e soddisfacimento di vincoli. Un approccio semplice ma diffuso è quello di utilizzare l’euristica del **margine minimo**: a ogni iterazione, si schedula per l’inizio il prima possibile qualsiasi azione non schedulata con tutti i suoi predecessori schedulati e con il margine minimo; poi si aggiornano i tempi *ES* e *LS* per ogni azione coinvolta e si ripete il processo. Questa euristica greedy assomiglia a quella MRV usata nel soddisfacimento di vincoli. Nella pratica funziona spesso bene, ma nel caso del nostro problema di assemblaggio restituisce una soluzione di 130 minuti, rispetto a quella da 115 della Figura 11.15.

margine minimo

Fin qui abbiamo assunto che l’insieme di azioni e vincoli di ordinamento fosse fissato. Sotto queste ipotesi, ogni problema di scheduling può essere risolto da una sequenza senza sovrapposizioni che eviti tutti i conflitti di risorse, purché ogni azione sia fattibile di per sé. Tuttavia, se un problema di scheduling risulta molto difficile, questa modalità di risoluzione potrebbe non essere ideale: potrebbe essere meglio riconsiderare le azioni e i vincoli per verificare se è possibile ottenere un problema molto più facile. Di conseguenza, ha senso *integrare* pianificazione e scheduling prendendo in considerazione la durata e le sovrapposizioni delle azioni già durante la costruzione di un piano. Molti degli algoritmi trattati nel Paragrafo 11.2 possono essere espansi per gestire questo tipo di informazione.

11.7 Analisi degli approcci alla pianificazione

La pianificazione combina le due aree principali dell'IA che abbiamo trattato fin qui: la *ricerca* e la *logica*. Un pianificatore infatti può essere considerato un programma che cerca una soluzione, ma anche uno che dimostra (in modo costruttivo) l'esistenza di una soluzione. Il fertile interscambio di idee tra le due aree ha consentito ai pianificatori di passare da problemi semplici, in cui il numero di azioni e stati era ridotto a una decina o poco più, ad applicazioni industriali del mondo reale con milioni di stati e migliaia di azioni.

La pianificazione è prima di tutto un esercizio nel controllo dell'esplosione combinatoria. Se in un dominio ci sono n proposizioni primitive, ci sono 2^n stati. Contro visioni pessimiste, l'identificazione di sottoproblemi indipendenti può rappresentare un'arma potente. Nel caso migliore, in cui il problema è perfettamente scomponibile, la velocità aumenta esponenzialmente. Tuttavia, la possibilità di scomporre il problema è impedita dalle interazioni negative tra le azioni. SATPLAN è in grado di rappresentare relazioni logiche tra sottoproblemi. La ricerca in avanti affronta il problema euristicamente tentando di trovare pattern (sottoinsiemi di proposizioni) che coprano i sottoproblemi indipendenti. Poiché questo approccio è euristico, può funzionare anche quando i sottoproblemi non sono completamente indipendenti.

Sfortunatamente non abbiamo ancora una chiara comprensione di quali tecniche funzionino meglio su determinati tipi di problemi. È certamente possibile che emergeranno tecniche nuove, forse in grado di fornire una sintesi tra rappresentazioni in logica del primo ordine altamente espansive e gerarchiche e le rappresentazioni proposizionali, molto efficienti, che dominano oggi. Stiamo vedendo apparire sistemi di pianificazione con **portafoglio**, in cui è disponibile una raccolta di algoritmi da applicare a qualsiasi problema dato. Questo può essere fatto in modo selettivo (il sistema classifica ogni nuovo problema in modo da scegliere l'algoritmo migliore per risolverlo) o in parallelo (tutti gli algoritmi sono eseguiti in modo concorrente, ognuno su una CPU diversa) o alternando gli algoritmi secondo uno schedule.

portafoglio

11.8 Riepilogo

In questo capitolo abbiamo descritto la rappresentazione PDDL di problemi di pianificazione classici ed estesi, e presentato diversi approcci algoritmici per la loro risoluzione. I punti da ricordare sono i seguenti.

- I sistemi di pianificazione sono algoritmi di risoluzione di problemi che operano su rappresentazioni fattorizzate esplicite degli stati e delle azioni. Queste rappresentazioni rendono possibile derivare euristiche efficaci e sviluppare algoritmi potenti e flessibili.
- Il linguaggio PDDL (*planning domain definition language*) descrive gli stati iniziale e obiettivo come congiunzioni di letterali e le azioni in termini di precondizioni ed effetti. Esistono estensioni per rappresentare tempo, risorse, percezioni, piani condizionali e piani gerarchici.
- La ricerca nello spazio degli stati può procedere in avanti (**progressione**) o all'indietro (**regressione**). Si possono derivare euristiche efficaci con l'ipotesi dell'indipendenza dei sotto-obiettivi o mediante diversi rilassamenti del problema di pianificazione.
- Tra gli altri approcci vi sono la codifica di un problema di pianificazione come problema di soddisfabilità booleana o come problema di soddisfacimento di vincoli e la ricerca esplicita nello spazio di piani parzialmente ordinati.
- La pianificazione basata su **reti gerarchiche** (HTN) permette all'agente di accettare i consigli del progettista del dominio, espressi sotto forma di **azioni di alto livello** (HLA) che possono essere implementate in vari modi da sequenze di azioni di livello inferiore. Gli effetti delle HLA possono essere definiti con una **semantica angelica**, che consente di derivare piani di alto livello di cui è possibile dimostrare la correttezza senza considerare le

implementazioni di livello inferiore. I metodi HTN sono in grado di creare i piani molto grandi richiesti dalle applicazioni reali.

- I **piani condizionali** permettono all'agente di percepire il mondo durante l'esecuzione e decidere quale ramo del piano seguire. In alcuni casi, la **pianificazione senza sensori o conformante** può consentire di costruire un piano che funziona senza necessità di ricevere percezioni durante l'esecuzione. Sia i piani conformanti sia quelli condizionali possono essere costruiti con una ricerca nello spazio degli **stati-credenza**. La rappresentazione e il calcolo efficiente di stati-credenza sono problemi fondamentali.
- Un **agente di pianificazione online** sfrutta il monitoraggio dell'esecuzione e introduce riparazioni nel piano quando è necessario per rimediare a situazioni inattese, che possono essere dovute ad azioni non deterministiche, eventi esogeni o modelli dell'ambiente sbagliati.
- Molte azioni consumano **risorse**, come denaro, benzina o materie prime. È conveniente trattare queste risorse come misure numeriche aggregate anziché cercare di ragionare su, per esempio, ogni singola moneta o banconota nel mondo. Il tempo è una delle risorse più importanti e può essere gestito da speciali algoritmi di scheduling; oppure lo scheduling può essere integrato con la pianificazione.
- Questo capitolo estende la pianificazione classica per coprire ambienti non deterministicci (in cui i risultati delle azioni sono incerti), ma non rappresenta l'ultima parola sulla pianificazione. Il Capitolo 17 descrive tecniche per ambienti stocastici (in cui i risultati delle azioni sono associati a probabilità): processi decisionali di Markov, processi decisionali di Markov parzialmente osservabili e teoria dei giochi. Nel Capitolo 22 del Volume 2 mostriamo che l'apprendimento per rinforzo consente a un agente di imparare come comportarsi in base a successi e fallimenti passati.

Note storiche e bibliografiche

La pianificazione in IA scaturì dagli studi sulla ricerca nello spazio degli stati, sulla dimostrazione di teoremi e sulla teoria del controllo. STRIPS (Fikes e Nilsson, 1971, 1993), il primo sistema di pianificazione importante, fu progettato come componente di pianificazione per il software del robot Shakey allo SRI. La prima versione del programma veniva eseguita su un computer dotato di soli 192 KB di memoria. La sua struttura di controllo generale fu modellata su quella di GPS, il General Problem Solver (Newell e Simon, 1961), un sistema di ricerca nello spazio degli stati che sfruttava l'analisi mezzi-finì.

Il linguaggio di rappresentazione STRIPS si è poi evoluto nell'Action Description Language, o ADL (Pednault, 1986) e poi nel Problem Domain Description Language, o PDDL (Ghallab *et al.*, 1998), che fu utilizzato come linguaggio standard per l'International Planning Competition a partire dal 1998. La versione più recente è PDDL 3.1 (Kovacs, 2011).

All'inizio degli anni 1970 i pianificatori scomponeranno i problemi calcolando un sottopiano per ogni sotto-objettivo e poi concatenando secondo qualche ordine tutti i sottopiani. Questo approccio, chiamato **pianificazione lineare** da Sacerdoti (1975), si rivelò presto incompleto: in effetti non può risolvere alcuni problemi molto semplici, come l'anomalia di Sussman (cfr. Esercizio 11.SUSS), scoperta da Allen Brown durante la sperimentazione con il sistema HACKER (Sussman, 1975). Un pianificatore completo deve consentire l'**interleaving**, ovvero l'alternanza di azioni di sottopiani diversi in una singola sequenza. Il sistema WARPLAN di Warren (1974) riuscì a fare ciò e dimostrò come il linguaggio di programmazione logica Prolog sia in grado di produrre programmi concisi; WARPLAN è lungo solamente 100 righe di codice.

La pianificazione con ordinamento parziale dominò i vent'anni successivi di ricerca, con lavori teorici che descrivevano il rilevamento di conflitti (Tate, 1975) e

la protezione di condizioni raggiunte (Sussman, 1975), e implementazioni come NOAH (Sacerdoti, 1977) e NONLIN (Tate, 1977). Si arrivò così a modelli formali (Chapman, 1987; McAllester e Rosenblitt, 1991) che consentivano l’analisi teorica di vari algoritmi e problemi di pianificazione, e a un sistema ampiamente distribuito, UCPOP (Penberthy e Weld, 1992).

Drew McDermott sospettò che l’interesse sulla pianificazione con ordinamento parziale tendesse a mettere in secondo piano altre tecniche che dovevano forse essere riconsiderate dato che i computer di quei tempi erano 100 volte più potenti di quelli in uso ai tempi di Shakey. Il suo UNPOP (McDermott, 1996) era un programma di pianificazione su spazio degli stati che utilizzava l’euristica che ignora le liste di eliminazioni. HSP, Heuristic Search Planner (Bonet e Geffner, 1999; Haslum, 2006) rese praticabile la ricerca nello spazio degli stati per grandi problemi di pianificazione. FF, o Fast Forward Planner (Hoffmann, 2001; Hoffmann e Nebel, 2001; Hoffmann, 2005), e la sua variante FASTDOWNWARD (Helmert, 2006) vinsero gare internazionali di pianificazione negli anni 2000.

Anche per la ricerca bidirezionale (cfr. Paragrafo 3.4.5) è nota la mancanza di euristiche, ma si è ottenuto qualche successo utilizzando la ricerca all’indietro per creare un **perimetro** attorno all’obiettivo e poi raffinando un’euristica per cercare in avanti verso tale perimetro (Torralba *et al.*, 2016). Il pianificatore con ricerca bidirezionale SYMBA* (Torralba *et al.*, 2016) ha vinto la gara del 2016.

Molti studiosi si sono interessati al PDDL e al paradigma della pianificazione per poter usare euristiche indipendenti dal dominio. Hoffmann (2005) analizza lo spazio di ricerca dell’euristica che ignora le liste di eliminazione. Edelkamp (2009) e Haslum *et al.* (2007) spiegano come costruire database di pattern per euristiche di pianificazione. Felner *et al.* (2004) riportano risultati incoraggianti usando database di pattern per rompicapi con tasselli a scorrimento, che possono essere considerati come un dominio di pianificazione, ma Hoffmann *et al.* (2006) mostrano alcune limitazioni dell’astrazione per problemi di pianificazione classica. (Rintanen, 2012) discute euristiche di selezione di variabili specifiche per problemi di pianificazione risolti come SAT.

Helmert *et al.* (2011) descrivono il sistema Fast Downward Stone Soup (FDSS), un pianificatore con portafoglio che, come nella favola della zuppa di sassi, ci invita a utilizzare quanti più algoritmi di pianificazione possibile. Il sistema mantiene un insieme di problemi di addestramento e per ciascuno di essi, e per

ogni algoritmo, registra il tempo di esecuzione e il costo per la soluzione del problema. Poi, quando si presenta un problema nuovo, il sistema utilizza l’esperienza del passato per decidere quali algoritmi provare, con quali limiti temporali, e utilizza la soluzione con il costo minimo. FDSS è risultato tra i vincitori nell’International Planning Competition del 2018 (Seipp and Röger, 2018). Seipp *et al.* (2015) descrivono un approccio di apprendimento automatico per apprendere automaticamente un buon portafoglio, dato un problema nuovo. Vallati *et al.* (2015) hanno fornito una panoramica sulla pianificazione con portafoglio. L’idea di utilizzare portafogli di algoritmi per problemi di ricerca combinatoria si deve a Gomes e Selman (2001).

Sistla e Godefroid (2004) hanno esaminato la riduzione simmetrica, mentre Godefroid (1990) ha trattato le euristiche per ordinamento parziale. Richter e Helmert (2009) hanno illustrato i guadagni di efficienza ottenuti con la potatura in avanti usando azioni preferite.

Blum e Furst (1997) hanno rivitalizzato il campo della pianificazione con il loro sistema Graphplan, che era di ordini di grandezza più veloce rispetto ai pianificatori con ordinamento parziale dell’epoca. Bryce e Kambhampati (2007) hanno fornito una panoramica sui grafi di pianificazione. L’uso del calcolo delle situazioni per la pianificazione è stato introdotto da John McCarthy (1963) e raffinato Ray Reiter (2001).

Kautz *et al.* (1996) esamarono vari modi per proposizionalizzare schemi di azione, trovando che le forme più compatte non portavano necessariamente ai tempi di soluzione più rapidi. Un’analisi sistematica fu condotta da Ernst *et al.* (1997) che svilupparono anche un “compilatore” automatico per generare rappresentazioni proposizionali da problemi rappresentati in PDDL. Il pianificatore BLACKBOX, che combina idee di Graphplan e SATPLAN, fu sviluppato da Kautz e Selman (1998). Tra i pianificatori basati sul soddisfacimento di vincoli vi furono CPLAN (van Beek e Chen 1999) e GP-CSP (Do e Kambhampati, 2003).

C’è stato molto interesse anche nella rappresentazione di un piano come **diagramma binario di decisione** (BDD, *binary decision diagram*), una struttura dati compatta per espressioni booleane ampiamente studiata nella comunità dedicata alla verifica dell’hardware (Clarke e Grumberg, 1987; McMillan, 1993). Esistono tecniche per dimostrare le proprietà dei diagrammi binari di decisione, tra cui la proprietà di essere una soluzione di un problema di pianificazione. Cimatti *et al.*

(1998) hanno presentato un pianificatore basato su questo approccio. Sono state usate anche altre rappresentazioni, come la programmazione intera (Vossen *et al.*, 2001).

Sono stati effettuati alcuni interessanti confronti tra i vari approcci alla pianificazione. Helmert (2001) ha analizzato diverse classi di problemi di pianificazione, mostrando che gli approcci basati sui vincoli come Graphplan e SATPLAN sono i migliori per i domini NP-difficili, mentre quelli basati sulla ricerca si comportano meglio nei domini in cui è possibile trovare soluzioni accettabili senza backtracking. Graphplan e SATPLAN si trovano in difficoltà nei domini con molti oggetti perché devono creare molte azioni. In alcuni casi il problema può essere rimandato o evitato generando le azioni proposizionalizzate dinamicamente, soltanto quando se ne presenta la necessità, anziché istanziarle tutte prima di cominciare la ricerca.

Il primo meccanismo per la pianificazione gerarchica fu una funzionalità del programma STRIPS per l'apprendimento di **macrops** – “macro-operatori” che consistono in una sequenza di passi primitivi (Fikes *et al.*, 1972). Il sistema ABSTRIPS (Sacerdoti, 1974) introdusse l'idea di una **gerarchia di astrazioni**, in cui la pianificazione ai livelli superiori aveva il permesso di ignorare le precondizioni delle azioni di livello più basso per derivare la struttura generale di un piano funzionante. La tesi di dottorato di Austin Tate (1975b) e il lavoro di Earl Sacerdoti (1977) svilupparono le idee fondamentali della pianificazione HTN. Erol, Hendler e Nau (1994, 1996) presentano un pianificatore basato su una scomposizione gerarchica completa oltre a una serie di studi di complessità sui pianificatori HTN puri. La nostra trattazione di HLA e semantica angelica si deve a Marthi *et al.* (2007, 2008).

Uno degli obiettivi della pianificazione gerarchica è stato il riuso delle esperienze precedenti sotto forma di piani generalizzati. La tecnica dell'**apprendimento basato sulle spiegazioni** è stata usata per generalizzare piani precedentemente calcolati in sistemi quali SOAR (Laird *et al.*, 1986) e PRODIGY (Carbonell *et al.*, 1989). Un approccio alternativo è memorizzare i piani precedentemente costruiti nella loro forma originale e riusarli per risolvere per analogia nuovi problemi simili. Questo è l'approccio adottato dalla branca nota come **pianificazione basata sui casi** (Carbonell, 1983; Alterman, 1988). Kambhampati (1994) sostiene che la pianificazione basata sui casi dovrebbe essere analizzata come una forma di pianificazione per raffinamento e presenta una base formale per la pianificazione basata sui casi con ordinamento parziale.

I primi pianificatori non avevano condizioni e cicli, ma alcuni potevano utilizzare la coercizione per formare piani conformanti. Il sistema NOAH di Sacerdoti risolse il problema delle “chiavi e scatole” (in cui il pianificatore conosce molto poco dello stato iniziale) utilizzando la coercizione. Mason (1993) sostiene che spesso le percezioni possono e devono essere evitate nella pianificazione robotica, e descrisse un piano privo di sensori capace di spostare uno strumento in una posizione specifica del tavolo per mezzo di una serie di azioni, *indipendentemente* dalla sua posizione iniziale.

Goldman e Boddy (1996) introdussero il termine **pianificazione conformante**, notando che i piani privi di percezioni sono spesso efficaci anche se l'agente possiede dei sensori. Il primo pianificatore conformante moderatamente efficiente fu Conformant Graphplan (CGP) di Smith e Weld (1998). Ferraris e Giunchiglia (2000) e Rintanen (1999) hanno sviluppato indipendentemente pianificatori conformanti basati su SATPLAN. Bonet e Geffner (2000) descrivono un pianificatore conformante basato sulla ricerca euristica nello spazio degli stati-credenza, ispirandosi a idee sviluppate inizialmente negli anni 1960 per i processi decisionali di Markov parzialmente osservabili, o POMDP (cfr. Capitolo 17).

Oggi sono tre i principali approcci alla pianificazione conformante. I primi due utilizzano la ricerca euristica nello spazio degli stati credenza: HSCP (Bertoli *et al.*, 2001a) utilizza diagrammi binari di decisione (BDD) per rappresentare gli stati-credenza, mentre Hoffmann e Brafman (2006) adottano l'approccio “pi-gro” di calcolare precondizioni e test obiettivo a richiesta utilizzando un risolutore SAT. Il terzo approccio, sostenuto principalmente da Jussi Rintanen (2007), formula l'intero problema di pianificazione senza sensori come una formula booleana quantificata (QBF) e lo risolve utilizzando un risolutore di QBF generico. Gli attuali pianificatori conformanti sono cinque ordini di grandezza più veloci di CGP. Il vincitore della gara di pianificazione conformante del 2006 all'International Planning Competition è stato T_0 (Palacios e Geffner, 2007), che utilizza la ricerca euristica nello spazio degli stati-credenza mantenendo una semplice rappresentazione dello stato-credenza mediante la definizione di letterali derivati che coprono effetti condizionali. Bryce e Kambhampati (2007) discutono come si possa generalizzare un grafo di pianificazione per generare buone euristiche per la pianificazione conformante e condizionale.

L'approccio alla pianificazione condizionale seguito in questo capitolo si basa su Hoffmann e Brafman

(2005) ed è stato influenzato dagli algoritmi di ricerca efficiente per grafi AND-OR ciclici sviluppati da Jimenez e Torras (2000) e da Hansen e Zilberstein (2001). Il problema della pianificazione condizionale ha ricevuto maggiore attenzione dopo la pubblicazione dell'importante articolo di Drew McDermott (1978a) *Planning and Acting*. Bertoli *et al.* (2001b) descrivono MBP (Model-Based Planner), che utilizza diagrammi di decisione binari per svolgere pianificazione conforme e condizionale. Alcuni autori utilizzano i termini “pianificazione condizionale” e “pianificazione di contingenza” come sinonimi, mentre altri utilizzano “condizionale” per riferirsi ad azioni con effetti non deterministici e “di contingenza” per riferirsi all’uso della percezione per superare la parziale osservabilità.

In retrospettiva, è possibile vedere come i più importanti algoritmi di pianificazione classica hanno stimolato lo sviluppo di versioni estese per domini incerti. La ricerca euristica in avanti nello spazio degli stati ha condotto alla ricerca in avanti nello spazio degli stati-credenza (Bonet e Geffner, 2000; Hoffmann e Brafman, 2005); SATPLAN ha dato origine a SATPLAN stocastico (Majercik e Littman, 2003) e alla pianificazione con logica di Boole quantificata (Rintanen, 2007); la pianificazione con ordinamento parziale ha portato a UWL (Etzioni *et al.*, 1992) e CNLP (Peot e Smith, 1992). Graphplan ha portato a Sensory Graphplan o SGP (Weld *et al.*, 1998).

Il primo pianificatore online con monitoraggio dell'esecuzione è stato PLANEX (Fikes *et al.*, 1972), che operava insieme al pianificatore STRIPS per controllare il robot Shakey. SIPE (System for Interactive Planning and Execution monitoring) (Wilkins, 1988, 1990) è stato il primo pianificatore a occuparsi sistematicamente del problema della ripianificazione. È stato usato in progetti dimostrativi in vari domini, tra cui le operazioni di pianificazione sul ponte di volo di una portaerei, il job-shop scheduling per una fabbrica di birra australiana e la pianificazione della costruzione di edifici multipiano (Kartam e Levitt, 1990).

A metà degli anni 1980 il pessimismo riguardo la scarsa velocità dei sistemi di pianificazione portò alla proposta di agenti reattivi denominati sistemi di **pianificazione reattiva** (Brooks, 1986; Agre e Chapman, 1987). I “piani universali” (Schoppers, 1987, 1989) furono sviluppati come un metodo tabellare per la pianificazione reattiva, ma risultarono essere una riscoperta dell'idea delle **politiche** usata da molto tempo nei processi decisionali di Markov (cfr. Capitolo 17). Koenig (2001) propone una rassegna delle tecniche di pianificazione online con il nome *Agent-Centred Search*.

La pianificazione con vincoli temporali fu affrontata per la prima volta da DEVISER (Vere, 1983). La rappresentazione del tempo nei piani fu trattata da Allen (1984) e da Dean *et al.* (1990) nel sistema FORBIN. NONLIN+ (Tate e Whiter, 1984) e SIPE (Wilkins, 1990) potevano ragionare sull'allocazione di risorse limitate ai vari passi di un piano. O-PLAN (Bell e Tate, 1985) è stato applicato nel software per la pianificazione degli acquisti in Price Waterhouse e alla pianificazione dell'assemblaggio delle automobili di Jaguar.

I due pianificatori SAPA (Do e Kambhampati, 2001) e T4 (Haslum e Geffner, 2001) usavano entrambi la ricerca in avanti nello spazio degli stati, con euristiche sofisticate per gestire la durata delle azioni e le risorse. Un'alternativa è usare linguaggi di descrizione delle azioni molto espressivi e guidarli con euristiche specifiche del dominio scritte da esperti umani, come fanno ASPEN (Fukunaga *et al.*, 1997), HSTS (Jonsson *et al.*, 2000) e IxTeT (Ghallab e Laruelle, 1994).

Sono stati realizzati numerosi sistemi ibridi di pianificazione e scheduling: ISIS (Fox *et al.*, 1982; Fox, 1990) è stato utilizzato per il job-shop scheduling presso Westinghouse, GARI (Descotte e Latombe, 1985) ha pianificato la lavorazione delle macchine per la costruzione di parti meccaniche, FORBIN è stato impiegato per il controllo di fabbrica, NONLIN+ è stato utilizzato per la pianificazione della logistica navale. Abbiamo scelto di presentare pianificazione e scheduling come due problemi separati; (Cushing *et al.*, 2007) mostra che questo può portare a incompletezza su certi problemi.

In ambito aerospaziale c'è una lunga tradizione di ricerca sullo scheduling. T-SCHED (Drabble, 1990) è stato usato per lo scheduling delle sequenze di comando per la missione del satellite UOSAT-II. OPTIMUM-AIV (Aarup *et al.*, 1994) e PLAN-ERS1 (Fuchs *et al.*, 1990), entrambi basati su O-PLAN, furono usati presso la European Space Agency rispettivamente per l'assemblaggio di veicoli spaziali e la pianificazione delle osservazioni. SPIKE (Johnston e Adorf, 1992) è stato usato alla NASA per la pianificazione delle osservazioni del telescopio spaziale Hubble, mentre lo Space Shuttle Ground Processing Scheduling System (Deale *et al.*, 1994) svolge il job-shop scheduling dei turni di lavoro di 16.000 persone. Remote Agent (Muscettola *et al.*, 1998) fu il primo pianificatore-scheduler autonomo a controllare un veicolo spaziale durante il volo a bordo della sonda Deep Space One nel 1999. Le applicazioni spaziali hanno guidato lo sviluppo di algoritmi per l'allocazione di risorse; cfr. Laborie (2003) e Muscettola (2002). La letteratura sullo scheduling è

presentata in un articolo ormai divenuto un classico (Lawler *et al.*, 1993), un libro (Pinedo, 2008) e un manuale (Blazewicz *et al.*, 2007).

La complessità computazionale della pianificazione è stata studiata da diversi autori (Bylander, 1994; Ghallab *et al.*, 2004; Rintanen, 2016). Ci sono due attività principali: **PlanSAT** è il problema di determinare se esiste un piano che risolva un problema di pianificazione; **Bounded PlanSAT** chiede se esiste una soluzione di lunghezza k o inferiore; lo si può utilizzare per trovare un piano ottimo. Entrambi sono problemi decidibili per la pianificazione classica (perché il numero di stati è finito), ma se aggiungiamo al linguaggio i simboli funzionali, allora il numero di stati diventa infinito e PlanSAT diventa solo semidecidibile. Per problemi proposizionalizzati, sia PlanSAT che Bounded PlanSAT rientrano nella classe di complessità PSPACE, una classe più grande (e quindi più difficile) di NP che si riferisce a problemi che possono essere risolti da una macchina di Turing deterministica con una quantità di spazio polinomiale. Questi risultati teorici sono scoraggianti, ma nella pratica i problemi che si presentano tendono a non essere così complessi. Il reale vantaggio del formalismo della pia-

nificazione classica è che ha facilitato lo sviluppo di euristiche indipendenti dal dominio e molto accurate, mentre altri approcci non sono stati altrettanto fruttuosi.

Readings in Planning (Allen *et al.*, 1990) è un'esaurente antologia dei primi lavori nel campo. Weld (1994, 1999) fornisce due panoramiche eccellenti degli algoritmi di pianificazione degli anni 1990. È interessante notare i cambiamenti avvenuti nei cinque anni che separano i due lavori: il primo si concentra sulla pianificazione con ordinamento parziale, il secondo introduce GRAPHPLAN e SATPLAN. *Automated Planning and Acting* (Ghallab *et al.*, 2016) è un eccellente manuale su tutti gli aspetti della pianificazione. Il testo *Planning Algorithms* di LaValle (2006) tratta la pianificazione classica e stocastica, esaminando in modo ampio la pianificazione del movimento dei robot.

La ricerca sulla pianificazione è stata una parte centrale dell'IA fin dall'inizio, e articoli che la riguardano si trovano in tutte le riviste e i congressi dell'IA. Ci sono anche congressi specializzati, come l'International Conference on Automated Planning and Scheduling e l'International Workshop on Planning and Scheduling for Space.

Conoscenza incerta e ragionamento in condizioni di incertezza

Capitolo 12 Quantificare l'incertezza

Capitolo 13 Ragionamento probabilistico

**Capitolo 14 Ragionamento probabilistico
nel tempo**

Capitolo 15 Programmazione probabilistica

Capitolo 16 Decisioni semplici

Capitolo 17 Decisioni complesse

Capitolo 18 Decisioni multiagente



Quantificare l'incertezza

- 12.1 Agire in condizioni di incertezza
- 12.2 Notazione base della teoria della probabilità
- 12.3 Inferenza basata su distribuzioni congiunte complete
- 12.4 Indipendenza
- 12.5 La regola di Bayes e il suo utilizzo
- 12.6 Modelli di Bayes ingenui
- 12.7 Il mondo del wumpus rivisitato
- 12.8 Riepilogo
Note storiche e bibliografiche

In cui vediamo come domare l'incertezza con gradi numerici di credenza.

12.1 Agire in condizioni di incertezza

Nel mondo reale gli agenti si trovano ad affrontare l'**incertezza**, che può essere dovuta a osservabilità parziale, non determinismo o presenza di avversari. Un agente potrebbe anche non sapere mai con sicurezza in quale stato si trovi ora o dove si troverà dopo una sequenza di azioni.

Abbiamo visto che gli agenti logici gestiscono l'incertezza tenendo traccia di uno **stato-credenza** – una rappresentazione dell'insieme di tutti gli stati possibili in cui potrebbe trovarsi – e generando un piano condizionale che gestisca ogni possibile eventualità che i suoi sensori possano riportare durante l'esecuzione. Questo approccio funziona per problemi semplici, ma presenta alcuni svantaggi:

- l'agente deve considerare *ogni possibile* spiegazione per le osservazioni dei suoi sensori, anche le più improbabili. In questo modo si genera uno stato-credenza enorme e pieno di possibilità poco realistiche;
- un piano condizionale corretto che gestisca ogni eventualità può crescere senza limiti e deve considerare contingenze del tutto improbabili;
- a volte non esiste un piano che dia garanzia di raggiungere l'obiettivo, tuttavia l'agente deve agire. Serve quindi un modo per confrontare la bontà di piani che non offrono garanzie.

Supponiamo per esempio che un taxi automatizzato abbia l'obiettivo di portare un passeggero all'aeroporto in tempo per prendere il volo. Il taxi costruisce un piano, A_{90} , che prevede di partire da casa 90 minuti prima della partenza del volo e guidare a una velocità ragionevole.

Anche se l’aeroporto dista solo una decina di chilometri, l’agente non sarà in grado di concludere con assoluta certezza che “il piano A_{90} ci farà giungere all’aeroporto in tempo”. Arriverà invece a una conclusione più debole: “il piano A_{90} ci farà arrivare all’aeroporto in tempo a patto che la macchina non abbia un guasto o finisca la benzina, che non capiti un incidente, che la strada non sia bloccata, che non ci colpisca un meteorite...”. Non è possibile dedurre con certezza alcuna di queste condizioni, perciò non possiamo inferire il successo del piano. Questo è il **problema della qualificazione** che abbiamo menzionato nel Capitolo 7, per cui finora non abbiamo visto alcuna soluzione reale.

Nondimeno, in un certo senso A_{90} è effettivamente la cosa giusta da fare. Ma che cosa vogliamo dire con questo? Come abbiamo visto nel Capitolo 2, vogliamo dire che, di tutti i piani che potrebbero essere eseguiti, A_{90} è quello che massimizza la misura di prestazione attesa dell’agente (dove l’attesa è relativa alla conoscenza che l’agente ha dell’ambiente). La misura di prestazione include: arrivare in tempo per il volo, evitare una lunga e noiosa attesa all’aeroporto e non prendere multe per eccesso di velocità lungo la strada. La conoscenza in possesso dell’agente non può garantire questi possibili esiti di A_{90} , ma può fornire un qualche grado di credenza sul loro raggiungimento. Piani alternativi, come A_{120} , possono aumentare la probabilità di arrivare all’aeroporto in tempo, ma aumentano anche il rischio di una lunga e noiosa attesa. *La cosa giusta da fare – la **decisione razionale** – dipende quindi sia dall’importanza relativa dei vari obiettivi, sia dalla probabilità e dalla misura del loro raggiungimento*. Nel seguito di questo paragrafo approfondiremo questi concetti, preparandoci così allo sviluppo delle teorie generali del ragionamento incerto e delle decisioni razionali, che esamineremo in questo capitolo e nei successivi.



12.1.1 Riassumere l’incertezza

Consideriamo un esempio di ragionamento incerto: diagnosticare il mal di denti di un paziente. La diagnosi – applicata alla medicina, alle riparazioni meccaniche o a qualsiasi altro campo – quasi immancabilmente prevede un certo grado di incertezza. Per mostrare l’inadeguatezza dell’approccio logico, cerchiamo di scrivere regole per le diagnosi dentarie usando la logica proposizionale. Considerate la regola seguente:

MalDiDenti \Rightarrow Carie .

Il problema è che questa regola è sbagliata. Non tutti i pazienti che accusano mal di denti hanno carie; alcuni potrebbero avere una gengivite, un ascesso o altri problemi:

MalDiDenti \Rightarrow Carie \vee Gengivite \vee Ascesso ...

Sfortunatamente, per rendere valida la regola dovremmo aggiungere una lista quasi infinita di possibili problematiche. Potremmo provare allora a trasformarla in una regola causale:

Carie \Rightarrow MalDiDenti .

Ma neppure questa regola è corretta, dato che non tutte le carie causano dolore. L’unico modo di correggere la regola è renderla logicamente esaustiva, arricchendo la parte sinistra con tutte le qualificazioni necessarie affinché una carie causi dolore. Cercare di usare la logica per gestire un dominio come la diagnosi medica fallisce quindi per tre ragioni principali.

pigrizia

ignoranza teorica

ignoranza pratica

- **Pigrizia:** elencare l’insieme completo di premesse e conseguenze che rendono una regola priva di eccezioni richiede troppo lavoro, e le regole risultano difficili da utilizzare.
- **Ignoranza teorica:** la scienza medica non ha una teoria completa per il dominio.
- **Ignoranza pratica:** anche se conosciamo tutte le regole, potremmo essere incerti nei confronti di un particolare paziente perché non sono stati eseguiti tutti gli esami necessari, magari per impossibilità di farlo.

La relazione tra mal di denti e carie, molto semplicemente, non è una conseguenza logica stretta in nessuna delle due direzioni. Questo è tipico del dominio medico, così come di molti altri in cui è richiesta la formulazione di giudizi: l'ambito legale, quello degli affari, del design, della riparazione di automobili, del giardinaggio, degli appuntamenti amorosi e così via. La conoscenza dell'agente può, nel caso migliore, fornire solo un **grado di credenza** nelle formule. Il nostro strumento principale per la gestione dei gradi di credenza è la **teoria della probabilità**. Nella terminologia del Paragrafo 8.1, gli **impegni ontologici** della logica e della teoria della probabilità sono identici – il mondo è composto di fatti che valgono o non valgono in ogni caso particolare – ma gli **impegni epistemologici** sono diversi: un agente logico crede che ogni formula sia vera o falsa oppure non ha opinione, mentre un agente probabilistico potrebbe avere un grado numerico di credenza compreso tra 0 (per formule certamente false) e 1 (per formule certamente vere).

La teoria della probabilità fornisce un modo per riassumere l'incertezza che deriva dalla nostra pigrizia e ignoranza, risolvendo così il problema della qualificazione. Potremmo anche non sapere con certezza che cosa affligge un particolare paziente, ma sappiamo che, nell'80% dei casi (poniamo) – cioè con probabilità di 0,8 – un paziente che ha mal di denti ha una carie. Questo significa che ci aspettiamo che, prendendo tutte le situazioni indistinguibili da quella corrente per quanto riguarda la conoscenza dell'agente, il paziente avrà una carie nell'80% di esse. Questa credenza potrebbe essere derivata da dati statistici (l'80% dei pazienti registrati con un mal di denti aveva una carie) o da qualche regola generale, o anche da una combinazione di fonti.

Un punto che può creare confusione è che nel mondo reale, al momento della diagnosi, non vi è incertezza: il paziente ha una carie oppure no. Che cosa significa, quindi, affermare che la probabilità che abbia una carie è 0,8? Non dovrebbe essere 0 o 1? La risposta è che le affermazioni probabilistiche sono fatte rispetto a uno stato di conoscenza, non rispetto al mondo reale. Affermiamo: “La probabilità che il paziente abbia una carie, *dato che ha mal di denti*, è 0,8”. Se in seguito veniamo a sapere che il paziente ha già avuto problemi gengivali, possiamo fare un'affermazione diversa: “La probabilità che il paziente abbia una carie, dato che ha mal di denti e ha già avuto problemi gengivali, è 0,4”. Se otteniamo ulteriori evidenze contro l'ipotesi della carie, possiamo affermare: “La probabilità che il paziente abbia una carie, dato tutto ciò che sappiamo, è quasi 0”. Notate che queste affermazioni non si contraddicono tra loro: ognuna è un'affermazione separata relativa a un diverso stato di conoscenza.



12.1.2 Incertezza e decisioni razionali

Consideriamo ancora il piano A_{90} per andare in aeroporto. Supponiamo che la sua probabilità di successo sia pari al 97%: questo significa che è la scelta razionale? Non necessariamente: potrebbero esistere altri piani, come A_{180} , con una probabilità di successo più alta. Se è assolutamente vitale non perdere il volo, potrebbe valere la pena di rischiare un'attesa più lunga all'aeroporto. E che dire di A_{1440} , un piano che prevede di partire alla volta dell'aeroporto con 24 ore di anticipo? Nella maggior parte dei casi non sarà una buona scelta dato che, pur garantendo la quasi certezza di imbarcarsi in tempo, richiederà un'attesa inaccettabile – per non parlare della possibilità di mangiare male all'aeroporto.

Per compiere scelte simili un agente deve prima di tutto avere **preferenze** sui possibili **esiti** dei vari piani. Un esito è la descrizione completa di uno stato, che specifica fattori come il fatto di arrivare in tempo per il volo e la durata dell'attesa all'aeroporto. Per rappresentare le preferenze e ragionare quantitativamente con esse useremo la **teoria dell'utilità**, che afferma che ogni stato (o sequenza di stati) ha un certo grado di utilità per l'agente, e che quest'ultimo preferirà sempre stati con utilità maggiore.

L'utilità di uno stato è relativa a un agente. Per esempio, l'utilità di uno stato in cui il Bianco vince una partita a scacchi è ovviamente molto alta per l'agente che gioca con i pezzi

grado di credenza
teoria della probabilità

preferenze
esiti

teoria dell'utilità

teoria delle decisioni**Massima Utilità Attesa (MEU)**

bianchi, ma bassa per il Nero. Non possiamo tuttavia considerare rigidamente i punteggi di 1, 1/2 e 0 imposti dalle regole dei tornei di scacchi, infatti alcuni giocatori (tra cui gli autori) sarebbero ben contenti di pareggiare contro il campione mondiale, laddove altri giocatori (come lo sfidante per il titolo) potrebbero non esserlo. Non c’è modo di valutare obiettivamente i gusti o le preferenze: potete pensare che un agente che preferisce il gelato al gusto di gomma da masticare al jalapeño rispetto a quello al cioccolato sia un po’ strano, ma non potete dire che è irrazionale. Una funzione di utilità può tenere conto di qualsiasi insieme di preferenze – strane o tipiche, nobili o perverse. Le utilità possono persino incorporare un comportamento altruistico, semplicemente includendo il benessere degli altri come uno dei fattori che contribuiscono al calcolo.

Le preferenze sono combinate con le probabilità nella teoria generale delle decisioni razionali chiamata **teoria delle decisioni**:

$$\text{teoria delle decisioni} = \text{teoria della probabilità} + \text{teoria dell'utilità} .$$

L’idea fondamentale della teoria delle decisioni è che *un agente è razionale se e solo se sceglie l’azione che porta alla più alta utilità attesa, calcolata sulla media di tutti i possibili esiti dell’azione*. Questo è il principio della **Massima Utilità Attesa** (MEU, *maximum expected utility*). In questo caso il termine “attesa” significa la “media”, o “media statistica” delle utilità degli esiti, pesata dalla probabilità dell’esito. Abbiamo già incontrato questo principio nel Capitolo 5, quando abbiamo accennato brevemente alle decisioni ottime nel gioco del backgammon. Si tratta in effetti di un principio assolutamente generale per le decisioni di singoli agenti.

La Figura 12.1 abbozza la struttura di un agente che sfrutta la teoria delle decisioni per selezionare le azioni. A livello astratto, è identico agli agenti descritti nei Capitoli 4 e 7 che mantengono uno stato-credenza corrispondente alla storia delle percezioni fino al momento attuale. Dato lo stato-credenza e una conoscenza degli effetti delle azioni, l’agente può formulare predizioni probabilistiche sugli esiti delle azioni e quindi selezionare l’azione con la massima utilità attesa.

In questo capitolo e nel prossimo ci concentreremo sul compito di rappresentare ed elaborare informazione probabilistica in generale. Il Capitolo 14 tratta i metodi specificatamente dedicati alla rappresentazione e all’aggiornamento dello stato-credenza e alla predizione dell’ambiente. Il Capitolo 15 esamina i modi di combinare la teoria della probabilità con linguaggi formali espressivi come la logica del primo ordine e i linguaggi di programmazione generici. Il Capitolo 16 approfondisce la teoria dell’utilità e il Capitolo 17 presenta alcuni algoritmi per pianificare sequenze di azioni in ambienti stocastici. Il Capitolo 18 tratta l’estensione di questi concetti ad ambienti multiagente.

Figura 12.1

Un agente basato sulla teoria delle decisioni che sceglie azioni razionali.

```
function AGENTE-TD(percezione) returns un’azione
  persistent: stato_credenza, credenze probabilistiche sullo stato corrente del mondo
               azione, l’azione dell’agente
  aggiorna stato_credenza sulla base di azione e percezione
  calcola le probabilità degli esiti delle azioni,
  date le loro descrizioni e lo stato_credenza corrente
  seleziona l’azione con la più alta utilità attesa,
  date le probabilità degli esiti e l’informazione sull’utilità
  return azione
```

12.2 Notazione base della teoria della probabilità

Affinché il nostro agente possa rappresentare e usare informazioni probabilistiche, è necessario un linguaggio formale, mentre il linguaggio della teoria della probabilità è tradizionalmente informale, scritto da matematici umani per altri matematici umani. L'Appendice A presenta un'introduzione alla teoria della probabilità, mentre qui adottiamo un approccio più adatto alle esigenze dell'IA, collegandolo ai concetti della logica formale.

12.2.1 A che cosa fanno riferimento le probabilità

Come le asserzioni logiche, anche quelle probabilistiche riguardano mondi possibili. Tuttavia, mentre le asserzioni logiche affermano quali mondi possibili sono strettamente esclusi (tutti quelli in cui l'asserzione è falsa), le asserzioni probabilistiche indicano quanto sono probabili i vari mondi. Nella teoria della probabilità, l'insieme di tutti i mondi possibili è chiamato **spazio campionario**. I mondi possibili sono *mutuamente esclusivi* ed *esaurienti*: due mondi possibili non possono verificarsi entrambi, e un mondo possibile deve necessariamente verificarsi. Per esempio, se stiamo per lanciare due dadi (distinguibili), ci sono 36 mondi possibili da considerare: (1,1), (1,2), ..., (6,6). La lettera greca Ω (omega maiuscola) è usata per riferirsi allo spazio campionario, mentre la lettera ω (omega minuscola) si riferisce a elementi dello spazio campionario, cioè particolari mondi possibili.

Un **modello di probabilità** completamente specificato associa una probabilità numerica $P(\omega)$ a ogni mondo possibile.¹ Gli assiomi di base della teoria della probabilità affermano che ogni mondo possibile ha una probabilità compresa tra 0 e 1 e che la probabilità totale dell'insieme dei mondi possibili è 1:

$$0 \leq P(\omega) \leq 1 \text{ per ogni } \omega \text{ e } \sum_{\omega \in \Omega} P(\omega) = 1. \quad (12.1)$$

Per esempio, se assumiamo che nessun dado sia truccato e che i lanci non interferiscano tra loro, allora ognuno dei mondi possibili (1,1), (1,2), ..., (6,6) ha probabilità 1/36. Se i dadi sono truccati, invece, alcuni mondi avranno probabilità maggiori di altri, ma la somma complessiva delle probabilità sarà sempre 1.

Solitamente le asserzioni e le interrogazioni di tipo probabilistico non riguardano alcuni particolari mondi possibili, ma insiemi di mondi. Per esempio, potremmo chiedere la probabilità di ottenere 11 lanciando due dadi, la probabilità di ottenere due numeri uguali, e così via. Nella teoria della probabilità questi insiemi sono detti **eventi**, un termine che abbiamo già ampiamente usato nel Capitolo 10 per indicare un concetto diverso. In logica, un insieme di mondi corrisponde a una **proposizione** di un linguaggio formale; nello specifico, per ogni proposizione, il corrispondente insieme contiene soltanto i mondi possibili in cui tale proposizione vale (quindi “evento” e “proposizione” significano praticamente la stessa cosa in questo contesto, con la differenza che le proposizioni sono espresse in un linguaggio formale). La probabilità associata a una proposizione è definita come la somma delle probabilità dei mondi in cui tale proposizione vale:

$$\text{Per qualsiasi proposizione } \phi, P(\phi) = \sum_{\omega \in \phi} P(\omega). \quad (12.2)$$

Per esempio, nel lancio di dadi non truccati abbiamo $P(\text{Totale} = 11) = P((5, 6)) + P((6, 5)) = 1/36 + 1/36 = 1/18$. Notate che la teoria della probabilità non richiede la conoscenza completa delle probabilità di ogni mondo possibile. Per esempio, se crediamo che i dadi siano

spazio campionario

**modello
di probabilità**

evento

¹ Per ora ipotizziamo un insieme discreto e numerabile di mondi. In presenza di un insieme continuo sorgono alcune complicazioni che non sono particolarmente rilevanti per l'IA.

probabilità non condizionata**probabilità a priori evidenza****probabilità condizionata****probabilità a posteriori****regola del prodotto**

truccati per produrre lo stesso numero, potremmo *asserire* che $P(\text{doppio}) = 1/4$ senza sapere se i dadi “preferiscono” il doppio 6 al doppio 2. Esattamente come avviene per le asserzioni logiche, questa asserzione *vincola* il modello di probabilità sottostante senza determinarlo completamente.

Le probabilità come $P(\text{Totale} = 11)$ e $P(\text{doppio})$ sono dette **non condizionate o a priori**; si riferiscono a gradi di credenza in proposizioni *in assenza di altre informazioni*. Nella maggior parte dei casi, tuttavia, abbiamo *alcune* informazioni, solitamente chiamate **evidenze** o prove, già note. Supponiamo per esempio che il primo dado sia stato già lanciato, mostrando 5, e siamo in attesa nervosa di vedere fermarsi l’altro che gira ancora. In questo caso non siamo interessati alla probabilità non condizionata di ottenere un numero doppio, ma alla probabilità **condizionata o a posteriori** di ottenere un numero doppio *dato che il primo dado lanciato è un 5*. Questa probabilità si scrive $P(\text{doppio} | \text{Dado}_1 = 5)$, dove il simbolo “|” si legge “dato”.²

In modo simile, se stiamo andando dal dentista per un controllo programmato, potremmo essere interessati alla probabilità a priori $P(\text{carie}) = 0,2$. Se invece andiamo dal dentista perché abbiamo mal di denti, ci interessa la probabilità condizionata $P(\text{carie} | \text{maldimenti}) = 0,6$.

È importante capire che $P(\text{carie}) = 0,2$ è ancora *valida* dopo l’osservazione di *maldimenti*; tuttavia, non è più particolarmente utile. Quando prende decisioni, un agente deve tenere conto di *tutte* le evidenze che ha osservato. È importante anche capire la differenza tra condizionamento e implicazione logica. L’asserzione che $P(\text{carie} | \text{maldimenti}) = 0,6$ non significa: “Ogni volta che *maldimenti* è vero, concludiamo che *carie* è vero con probabilità 0,6”, ma piuttosto: “Ogni volta che *maldimenti* è vero e *non abbiamo altre informazioni*, concludiamo che *carie* è vero con probabilità 0,6”. La condizione in più è importante; per esempio, se avessimo l’ulteriore informazione che il dentista non ha trovato carie, non concluderemmo certamente che *carie* è vero con probabilità 0,6; dovremmo invece usare $P(\text{carie} | \text{maldimenti} \wedge \neg \text{carie}) = 0$.

In termini matematici, le probabilità condizionate sono definite in termini di probabilità non condizionate come segue: per due proposizioni qualsiasi *a* e *b*, abbiamo

$$P(a | b) = \frac{P(a \wedge b)}{P(b)}, \quad (12.3)$$

che vale ogni volta che $P(b) > 0$. Per esempio,

$$P(\text{doppio} | \text{Dado}_1 = 5) = \frac{P(\text{doppio} \wedge \text{Dado}_1 = 5)}{P(\text{Dado}_1 = 5)}.$$

La definizione appare chiara se ricordate che osservando *b* si escludono tutti i mondi possibili in cui *b* è falsa, per cui rimane un insieme la cui probabilità totale è $P(b)$. All’interno di tale insieme, i mondi in cui *a* è vera devono soddisfare *a* \wedge *b* e costituiscono una frazione $P(a \wedge b)/P(b)$.

La definizione di probabilità condizionata data nell’Equazione (12.3) può essere scritta in forma diversa, chiamata **regola del prodotto**:

$$P(a \wedge b) = P(a | b)P(b). \quad (12.4)$$

La regola del prodotto è forse più facile da ricordare: deriva dal fatto che, affinché *a* e *b* siano vere, occorre che *b* sia vera e anche che *a* sia vera data *b*.

² Notate che la precedenza di “|” è tale che qualsiasi espressione della forma $P(\dots | \dots)$ significa sempre $P((\dots) | (\dots))$.

12.2.2 Il linguaggio delle proposizioni nelle asserzioni probabilistiche

In questo capitolo, e nel successivo, scriveremo le proposizioni che descrivono insiemi di mondi possibili utilizzando una notazione che combina elementi di logica proposizionale e soddisfacimento di vincoli. Nella terminologia del Paragrafo 2.4.7 si tratta di una **rappresentazione fattorizzata**, in cui un mondo possibile è rappresentato da un insieme di coppie variabile/valore. È anche possibile utilizzare una **rappresentazione strutturata**, maggiormente espressiva, come vedremo nel Capitolo 15.

Le variabili nella teoria della probabilità sono chiamate **variabili casuali** e i loro nomi iniziano con una lettera maiuscola. Nell'esempio dei dadi, quindi, *Totale* e *Dado₁* sono variabili casuali. Ogni variabile casuale è una funzione dal dominio dei mondi possibili Ω a un certo **intervallo**, l'insieme dei valori che la variabile può assumere. L'intervallo di *Totale* per due dadi è l'insieme {2,..., 12} e l'intervallo di *Dado₁* è {1,..., 6}. I nomi dei valori sono sempre in minuscolo, perciò potremmo scrivere $\sum_x P(X = x)$ per sommare sui valori di X . Una variabile casuale booleana ha l'intervallo {true, false}. Per esempio, la proposizione che descrive un doppio numero ottenuto ai dadi può essere scritta come *Doppio = true* (un intervallo alternativo per le variabili booleane è l'insieme {0, 1}, nel qual caso si dice che la variabile ha una distribuzione di **Bernoulli**). Per convenzione, le proposizioni della forma $A = \text{true}$ sono abbreviate in a , mentre $A = \text{false}$ è abbreviata in $\neg a$ (*doppio*, *carie* e *maldimenti* utilizzati nel precedente paragrafo sono abbreviazioni di questo tipo).

variabile casuale

intervallo

Bernoulli

Gli intervalli possono essere insiemi di token arbitrari. Potremmo scegliere come intervallo di *Età* l'insieme {giovane, adolescente, adulto} e come intervallo di *CondizioniAtmosferiche* l'insieme {sole, pioggia, coperto, neve}. Quando non c'è possibilità di ambiguità, spesso si utilizza un valore per rappresentare la proposizione che una particolare variabile ha tale valore; quindi, *sole* può stare per *CondizioniAtmosferiche = sole*.³

Negli esempi precedenti ci sono sempre intervalli finiti, ma le variabili possono anche avere intervalli infiniti, discreti (come gli interi) o continui (come i numeri reali). Se una variabile ha un intervallo ordinato, sono possibili anche le diseguaglianze, come *NumerodiAtomiInUniverso* $\geq 10^{70}$.

Infine, possiamo combinare queste proposizioni elementari (incluse le forme abbreviate delle variabili booleane) usando i connettivi della logica proposizionale. Per esempio, possiamo esprimere “La probabilità che la paziente abbia una carie, dato che è un'adolescente senza mal di denti, è 0,1” come segue:

$$P(\text{carie} | \neg \text{maldimenti} \wedge \text{adolescente}) = 0,1.$$

Nella notazione della probabilità, si utilizza comunemente una virgola per indicare la congiunzione, perciò potremmo scrivere $P(\text{carie} | \neg \text{maldimenti}, \text{adolescente})$.

A volte vorremo parlare delle probabilità di tutti i valori possibili di una variabile casuale. Potremmo scrivere:

$$\begin{aligned} P(\text{CondizioniAtmosferiche} = \text{sole}) &= 0,6 \\ P(\text{CondizioniAtmosferiche} = \text{pioggia}) &= 0,1 \\ P(\text{CondizioniAtmosferiche} = \text{coperto}) &= 0,29 \\ P(\text{CondizioniAtmosferiche} = \text{neve}) &= 0,01 , \end{aligned}$$

ma con un'abbreviazione potremo scrivere

$$\mathbf{P}(\text{CondizioniAtmosferiche}) = (0,6, 0,1, 0,29, 0,01),$$

³ Queste convenzioni nel loro insieme portano a una potenziale ambiguità nella notazione quando si effettua la somma sui valori di una variabile booleana: $P(a)$ è la probabilità che A sia *true*, mentre nell'espressione $\sum a P(a)$ indica semplicemente la probabilità di uno dei valori a di A .

distribuzione di probabilità

distribuzione categoriale

funzione di densità di probabilità

distribuzione di probabilità congiunta

dove la **P** in grassetto indica che il risultato è un vettore di numeri e ipotizziamo che vi sia un ordinamento predefinito (*sole, pioggia, coperto, neve*) sull’intervallo di *CondizioniAtmosferiche*. Diciamo che l’affermazione **P** definisce una **distribuzione di probabilità** per la variabile casuale *CondizioniAtmosferiche*, cioè un assegnamento di una probabilità a ogni possibile valore della variabile casuale (in questo caso, con un intervallo finito e discreto, la distribuzione è detta **distribuzione categoriale**). La notazione **P** è usata anche per le distribuzioni condizionate: **P**($X | Y$) fornisce i valori di $P(X = x_i | Y = y_j)$ per ogni possibile coppia i, j .

Per le variabili continue non è possibile scrivere l’intera distribuzione come vettore, perché i valori possibili sono infiniti. Possiamo però definire la probabilità che una variabile casuale assuma un valore x come funzione parametrizzata di x , generalmente chiamata **funzione di densità di probabilità**. Per esempio, la formula

$$P(\text{TemperaturaMezzogiorno} = x) = \text{Uniform}(x; 18C, 26C)$$

esprime la credenza che la temperatura a mezzogiorno sia distribuita uniformemente tra 18 e 26 gradi Celsius.

Le funzioni di densità di probabilità (a volte dette **PDF**, *probability density functions*) hanno un significato diverso dalle distribuzioni discrete. Dire che la densità di probabilità è uniforme da 18C a 26C significa che vi è una possibilità del 100% che la temperatura ricada in quell’intervallo esteso su 8C e una possibilità del 50% che ricada in un sottointervallo di 4C, e così via. Scriviamo la densità di probabilità per una variabile casuale continua X nel valore x come $P(X = x)$ o semplicemente $P(x)$; la definizione intuitiva di $P(x)$ è la probabilità che X ricada in un intorno arbitrariamente piccolo di x , divisa per l’ampiezza dell’intorno:

$$P(x) = \lim_{dx \rightarrow 0} P(x \leq X \leq x + dx)/dx.$$

Per *TemperaturaMezzogiorno* abbiamo

$$P(\text{TemperaturaMezzogiorno} = x) = \text{Uniform}(x; 18C, 26C) = \begin{cases} \frac{1}{8C} & \text{se } 18C \leq x \leq 26C \\ 0 & \text{altrimenti} \end{cases}$$

dove C sta per centigrado (non per una costante). In $P(\text{TemperaturaMezzogiorno} = 20,18C) = \frac{1}{8C}$, notate che $\frac{1}{8C}$ non è una probabilità ma una densità di probabilità. La probabilità che *TemperaturaMezzogiorno* sia esattamente 20,18C è zero, perché 20,18C è un intervallo di ampiezza 0. Alcuni autori usano simboli diversi per probabilità discrete e densità di probabilità; noi utilizziamo in entrambi i casi P per valori di probabilità specifici e **P** per vettori di valori, poiché è abbastanza raro che si verifichi confusione e le equazioni solitamente sono identiche. Notate che i valori di probabilità sono numeri senza unità, mentre le funzioni di densità hanno un’unità di misura, in questo caso il reciproco del grado centigrado. Se lo stesso intervallo di temperatura fosse espresso in gradi Fahrenheit, avrebbe un’ampiezza di 14,4 gradi e la densità sarebbe 1/14,4F.

Ci serve una notazione non solo per distribuzioni su singole variabili, ma anche per distribuzioni su variabili multiple. A questo scopo utilizziamo le virgolette. Per esempio, **P**(*CondizioniAtmosferiche, Carie*) denota le probabilità di tutte le combinazioni dei valori di *CondizioniAtmosferiche* e *Carie*. Si tratta di una tabella di probabilità 4×2 chiamata **distribuzione di probabilità congiunta** di *CondizioniAtmosferiche* e *Carie*. Possiamo anche mescolare variabili e valori specifici: **P**(*sole, Carie*) sarebbe un vettore di due elementi che fornisce le probabilità di avere una carie con una giornata soleggiata e nessuna carie con una giornata soleggiata.

Con la notazione **P** alcune espressioni sono molto più concise. Per esempio, le regole del prodotto (cfr. Equazione (12.4)) per tutti i valori possibili di *CondizioniAtmosferiche* e *Carie* si possono scrivere con un’unica equazione:

$$\mathbf{P}(\text{CondizioniAtmosferiche}, \text{Carie}) = \mathbf{P}(\text{CondizioniAtmosferiche} | \text{Carie})\mathbf{P}(\text{Carie}),$$

al posto delle seguenti $4 \times 2 = 8$ equazioni (scritte usando le abbreviazioni TA e C):

$$\begin{aligned} P(TA = \text{sole} \wedge C = \text{true}) &= P(TA = \text{sole}|C = \text{true}) P(C = \text{true}) \\ P(TA = \text{pioggia} \wedge C = \text{true}) &= P(TA = \text{pioggia}|C = \text{true}) P(C = \text{true}) \\ P(TA = \text{coperto} \wedge C = \text{true}) &= P(TA = \text{coperto}|C = \text{true}) P(C = \text{true}) \\ P(TA = \text{neve} \wedge C = \text{true}) &= P(TA = \text{neve}|C = \text{true}) P(C = \text{true}) \\ P(TA = \text{sole} \wedge C = \text{false}) &= P(TA = \text{sole}|C = \text{false}) P(C = \text{false}) \\ P(TA = \text{pioggia} \wedge C = \text{false}) &= P(TA = \text{pioggia}|C = \text{false}) P(C = \text{false}) \\ P(TA = \text{coperto} \wedge C = \text{false}) &= P(TA = \text{coperto}|C = \text{false}) P(C = \text{false}) \\ P(TA = \text{neve} \wedge C = \text{false}) &= P(TA = \text{neve}|C = \text{false}) P(C = \text{false}). \end{aligned}$$

Un caso degenere è quello di $\mathbf{P}(\text{sole}, \text{carie})$ che non ha variabili e quindi è un vettore a zero dimensioni, che possiamo considerare come un valore scalare.

Abbiamo così definito una sintassi per proposizioni e asserzioni probabilistiche e abbiamo fornito parte della semantica: l'Equazione (12.2) definisce la probabilità di una proposizione come la somma delle probabilità dei mondi in cui tale proposizione vale. Per completare la semantica dobbiamo dire quali sono i mondi e come determinare se una proposizione vale in un mondo. Riprendiamo questa parte direttamente dalla semantica della logica proposizionale, come segue: *un mondo possibile è definito come un assegnamento di valori a tutte le variabili casuali oggetto di considerazione*.



È facile notare che questa definizione soddisfa il requisito base che i mondi possibili siano mutuamente esclusivi ed esaustivi (Esercizio 12.EXEX). Per esempio, se le variabili casuali sono *Carie*, *Maldimenti* e *TempoAtmosferico*, allora ci sono $2 \times 2 \times 4 = 16$ mondi possibili. Inoltre, il valore di verità di una qualsiasi proposizione può essere determinato facilmente in tali mondi utilizzando lo stesso calcolo di verità ricorsivo che abbiamo usato per la logica proposizionale (cfr. Paragrafo 7.4.2).

Notate che alcune variabili casuali potrebbero essere ridondanti, nel senso che i loro valori sono ottenibili in tutti i casi dai valori di altre variabili. Per esempio, la variabile *Doppi* nel mondo dei due dati è vera esattamente quando $Dado_1 = Dado_2$. Se si include *Doppi* tra le variabili casuali in aggiunta a $Dado_1$ e $Dado_2$, sembra che il numero dei mondi possibili aumenti da 36 a 72, ma esattamente la metà di quei 72 mondi sarà logicamente impossibile e avrà probabilità 0. Dalla precedente definizione di mondi possibili segue che un modello di probabilità è completamente determinato dalla distribuzione congiunta di tutte le variabili casuali, la cosiddetta **distribuzione di probabilità congiunta completa**. Per esempio, dati *Carie*, *Maldimenti* e *CondizioniAtmosferiche*, la distribuzione congiunta completa è $\mathbf{P}(\text{Carie}, \text{Maldimenti}, \text{CondizioniAtmosferiche})$. Questa distribuzione congiunta può essere rappresentata come una tabella $2 \times 2 \times 4$ con 16 elementi. Poiché la probabilità di ogni proposizione è una somma su mondi possibili, una distribuzione congiunta completa è sufficiente, in linea di principio, per calcolare la probabilità di qualsiasi proposizione. Vedremo esempi di come fare questo nel Paragrafo 12.3.

**distribuzione
di probabilità
congiunta completa**

12.2.3 Gli assiomi della teoria della probabilità e la loro ragionevolezza

Gli assiomi di base della teoria della probabilità (Equazioni (12.1) e (12.2)) implicano determinate relazioni tra i gradi di credenza che si possono accordare a proposizioni logicamente correlate. Per esempio, possiamo ricavare la familiare relazione tra la probabilità di una proposizione e la probabilità della sua negazione:

$$\begin{aligned} P(\neg a) &= \sum_{\omega \in \neg a} P(\omega) && \text{per l'Equazione (12.2)} \\ &= \sum_{\omega \in \neg a} P(\omega) + \sum_{\omega \in a} P(\omega) - \sum_{\omega \in a} P(\omega) \\ &= \sum_{\omega \in \Omega} P(\omega) - \sum_{\omega \in a} P(\omega) && \text{raggruppando i primi due termini} \\ &= 1 - P(a) && \text{per (12.1) e (12.2).} \end{aligned}$$

principio di inclusione-esclusione

assiomi di Kolmogorov

Possiamo anche ricavare la ben nota formula della probabilità di una disgiunzione, talvolta chiamata **principio di inclusione-esclusione**:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b). \quad (12.5)$$

Questa regola si ricorda facilmente osservando che i casi in cui a è vera, insieme ai casi in cui b è vera, coprono certamente tutti i casi in cui è vera $a \vee b$; ma sommando i due insiemi di casi si conta due volte la loro intersezione, perciò dobbiamo sottrarre $P(a \wedge b)$.

Le Equazioni (12.1) e (12.5) sono spesso chiamate **assiomi di Kolmogorov** in onore del matematico Andrei Kolmogorov, che mostrò come costruire la teoria della probabilità a partire da questa semplice base e come gestire le difficoltà causate dalle variabili continue.⁴ Mentre l’Equazione (12.2) è in pratica una definizione, l’Equazione (12.5) rivela che gli assiomi in realtà vincolano i gradi di credenza che un agente può avere in riferimento a proposizioni in relazione logica tra loro, analogamente al fatto che un agente logico non può credere simultaneamente in A , B e $\neg(A \wedge B)$ perché non esiste un mondo possibile in cui sono vere tutte e tre. Con le probabilità, invece, le affermazioni non si riferiscono direttamente al mondo, ma allo stato di conoscenza dell’agente. E allora perché un agente può non ritener vere le seguenti credenze (che violano gli assiomi di Kolmogorov)?

$$P(a) = 0,4 \quad P(b) = 0,3 \quad P(a \wedge b) = 0,0 \quad P(a \vee b) = 0,8. \quad (12.6)$$

Questo tipo di domanda è stato oggetto di decenni di intenso dibattito tra coloro che sostengono l’uso delle probabilità come unica forma per trattare i gradi di credenza e coloro che sostengono approcci alternativi.

Un’argomentazione a favore degli assiomi della probabilità, enunciata per la prima volta da Bruno de Finetti nel 1931, è la seguente: se un agente ha un certo grado di credenza in una proposizione a , allora dovrebbe essere in grado di definire le quote a cui è indifferente scommettere a favore o contro a .⁵ Pensatelo come un gioco tra due agenti: l’Agente 1 afferma: “Il mio grado di credenza nell’evento a è 0,4”; allora l’Agente 2 è libero di scegliere se scommettere a favore o contro a a quote coerenti con il grado di credenza definito. In pratica, l’Agente 2 potrebbe scegliere di accettare la scommessa dell’Agente 1 che a si verificherà, offrendo 6 euro contro la puntata di 4 euro dell’Agente 1. Oppure l’Agente 2 potrebbe accettare la scommessa dell’Agente 1 che si verificherà $\neg a$, offrendo 4 euro contro i 6 euro dell’Agente 1. A questo punto osserviamo l’esito di a e chi ha ragione vincerà i soldi. Se i gradi di credenza di un agente non riflettono accuratamente il mondo, ci si attende che tale agente tenda a perdere denaro nel lungo periodo, rispetto a un agente le cui credenze riflettono meglio lo stato del mondo.

Il teorema di de Finetti non riguarda la scelta dei valori giusti per singole probabilità, ma la scelta dei valori per le probabilità di proposizioni in relazione logica tra loro: *se l’Agente 1 esprime un insieme di gradi di credenza che violano gli assiomi della teoria della probabilità, allora ci sarà una combinazione di puntate dell’Agente 2 che garantisce che Agente 1 perderà denaro ogni volta*. Per esempio, supponendo che l’Agente 1 abbia l’insieme di gradi di credenza dell’Equazione (12.6), la Figura 12.2 mostra che se l’Agente 2 sceglie di puntare 4 euro su a , 3 euro su b e 2 euro su $\neg(a \vee b)$, l’Agente 1 perderà sempre denaro, indipendentemente dagli esiti di a e b . Il teorema di de Finetti implica che nessun agente razionale può avere credenze che violano gli assiomi della probabilità.



⁴ Le difficoltà includono l’**insieme di Vitali**, un sottoinsieme ben definito dell’intervallo $[0, 1]$ con dimensione non ben definita.

⁵ Si potrebbe argomentare che le preferenze dell’agente riguardo il saldo del conto in banca siano tali che la possibilità di perdere 1 euro non è controbilanciata da una uguale possibilità di vincere 1 euro. Una possibile risposta consiste nel fare puntate abbastanza piccole da evitare questo problema. L’analisi di Savage (1954) elude del tutto il problema.

Proposizione	Credenza dell'Agente 1	Puntate dell'Agente 2	Puntate dell'Agente 1	Risultato per l'Agente 1 per ogni esito			
				a, b	$a, \neg b$	$\neg a, b$	$\neg a, \neg b$
a	0,4	€4 su a	€6 su $\neg a$	-€6	-€6	€4	€4
b	0,3	€3 su b	€7 su $\neg b$	-€7	€3	-€7	€3
$a \vee b$	0,8	€2 su $\neg(a \vee b)$	€8 su $a \vee b$	€2	€2	€2	-€8
				-€11	-€1	-€1	-€1

Figura 12.2 Poiché l'Agente 1 ha credenze inconsistenti, l'Agente 2 è in grado di elaborare un insieme di tre puntate che garantiscono una perdita per l'Agente 1 indipendentemente dal risultato di a e b .

Un'obiezione comune al teorema di de Finetti è che questo gioco a scommesse è piuttosto limitato. Per esempio, che succede se un agente rifiuta di scommettere? Termina subito il ragionamento? La risposta è che il gioco a scommesse è un modello astratto per situazioni decisionali in cui ogni agente è *inevitabilmente* coinvolto in ogni istante. Ogni azione (inclusa l'inazione) è un tipo di scommessa, e ogni esito può essere visto come una vittoria o perdita. Rifiutare di scommettere è come rifiutare di lasciare che il tempo scorra.

In favore dell'uso della probabilità sono state addotte molte altre motivazioni filosofiche, tra cui le più note sono quelle di Cox (1946), Carnap (1950) e Jaynes (2003), ognuna delle quali costruisce un insieme di assiomi per ragionare con gradi di credenza: nessuna contraddizione, corrispondenza con la logica classica (per esempio, se la credenza in A aumenta, allora la credenza in $\neg A$ deve diminuire), e così via. L'unico assioma controverso è quello per cui i gradi di credenza devono essere numeri, o almeno comportarsi come tali nel senso che devono essere transitivi (se la credenza in A è maggiore della credenza in B , e quest'ultima è maggiore della credenza in C , allora la credenza in A deve essere maggiore della credenza in C) e confrontabili (la credenza in A deve essere uguale a, maggiore o minore della credenza in B). Si può dimostrare che la probabilità è l'unico approccio che soddisfa questi assiomi.

Per come va il mondo, comunque, le dimostrazioni pratiche sono spesso più efficaci di quelle teoriche. Il successo dei sistemi di ragionamento basati sulla teoria della probabilità è stato molto più efficace delle argomentazioni filosofiche nell'attirare proseliti. Nel seguito vediamo come gli assiomi si possano sfruttare per effettuare inferenze.

12.3 Inferenza basata su distribuzioni congiunte complete

In questo paragrafo descriviamo un semplice metodo per l'**inferenza probabilistica**, ovvero il calcolo delle probabilità a posteriori di proposizioni di **interrogazione** date le evidenze osservate. Useremo la distribuzione congiunta completa come “base di conoscenza” da cui poter derivare le risposte a tutte le domande. Durante la trattazione presenteremo anche diverse tecniche utili per la manipolazione di equazioni che riguardano le probabilità.

inferenza
probabilistica
interrogazione

Cominciamo con un esempio molto semplice: un dominio che consiste nelle tre sole variabili booleane *MalDiDenti*, *Carie* e *Prende* (che indica se il temibile strumento d'acciaio appuntito del dentista “prende” nel mio dente, rilevando una cavità). La distribuzione di probabilità congiunta completa è la tabella $2 \times 2 \times 2$ riportata nella Figura 12.3.

Notate che la somma delle probabilità della distribuzione congiunta è pari a 1, come richiesto dagli assiomi. Notate anche che l'Equazione (12.2) ci fornisce un metodo diretto per calcolare la probabilità di ogni proposizione, semplice o complessa: basta semplicemente identificare i mondi possibili in cui la proposizione è vera e sommare le loro probabilità. Per esempio, ci sono sei mondi possibili in cui *carie* \vee *malldidenti* è verificata:

$$P(\text{carie} \vee \text{malldent}) = 0,108 + 0,012 + 0,072 + 0,008 + 0,016 + 0,064 = 0,28 .$$

Figura 12.3

Una distribuzione congiunta completa per il mondo *MalDiDenti, Carie, Prende*.

	maldidenti		\negmaldidenti	
	<i>prende</i>	\neg <i>prende</i>	<i>prende</i>	\neg <i>prende</i>
<i>carie</i>	0,108	0,012	0,072	0,008
\neg <i>carie</i>	0,016	0,064	0,144	0,576

probabilità marginale

marginalizzazione

Un’attività particolarmente comune è l’estrazione della distribuzione su un sottoinsieme particolare di variabili o una variabile singola. Per esempio, sommare tutti i valori della prima riga ci fornisce la **probabilità marginale**⁶ o non condizionata di *carie*:

$$P(\text{carie}) = 0,108 + 0,012 + 0,072 + 0,008 = 0,2.$$

Questo processo viene chiamato **marginalizzazione** o, con un termine inglese, **summing out** – perché sommiamo le probabilità per ogni valore possibile delle altre variabili, rimuovendole così dall’equazione. Possiamo scrivere la seguente regola generale di marginalizzazione per qualsiasi coppia di insiemi di variabili **Y** e **Z**:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{Z} = \mathbf{z}) \quad (12.7)$$

Dove $\Sigma_{\mathbf{z}}$ indica la somma su tutte le possibili combinazioni di valori dell’insieme di variabili **Z**. Come di consueto, possiamo abbreviare $\mathbf{P}(\mathbf{Y}, \mathbf{Z} = \mathbf{z})$ in questa equazione con $\mathbf{P}(\mathbf{Y}, \mathbf{z})$. Per l’esempio sulla carie, l’Equazione (12.7) corrisponde alla seguente equazione:

$$\begin{aligned} \mathbf{P}(\text{Carie}) &= \mathbf{P}(\text{Carie, maldidenti, prende}) + \mathbf{P}(\text{Carie, maldidenti, } \neg\text{prende}) \\ &\quad + \mathbf{P}(\text{Carie, } \neg\text{maldidenti, prende}) + \mathbf{P}(\text{Carie, } \neg\text{maldidenti, } \neg\text{prende}) \\ &\quad + \langle 0,108, 0,016 \rangle + \langle 0,012, 0,064 \rangle + \langle 0,072, 0,144 \rangle + \langle 0,008, 0,576 \rangle \\ &= \langle 0,2, 0,8 \rangle. \end{aligned}$$

condizionamento

Usando la regola del prodotto (Equazione (12.4)) possiamo sostituire $\mathbf{P}(\mathbf{Y}, \mathbf{z})$ nell’Equazione (12.7) con $\mathbf{P}(\mathbf{Y}|\mathbf{z})P(\mathbf{z})$, ottenendo la regola del **condizionamento**:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} | \mathbf{Z}) P(\mathbf{z}). \quad (12.8)$$

La marginalizzazione e il condizionamento risultano utili per molti tipi di derivazioni che coinvolgono espressioni probabilistiche.

Nella maggior parte dei casi ci interessa calcolare la probabilità *condizionata* di qualche variabile, avendo a disposizione altre variabili come evidenza. Le probabilità condizionate possono essere ricavate usando prima di tutto l’Equazione (12.3) per ottenere un’espressione che contiene probabilità non condizionate e successivamente valutando tale espressione per mezzo della distribuzione congiunta completa. Possiamo per esempio calcolare la probabilità di una carie, data l’evidenza rappresentata dalla presenza di mal di denti, nel modo che segue:

$$\begin{aligned} P(\text{carie} | \text{maldidenti}) &= \frac{P(\text{carie} \wedge \text{maldidenti})}{P(\text{maldidenti})} \\ &= \frac{0,108 + 0,012}{0,108 + 0,012 + 0,016 + 0,064} = 0,6. \end{aligned}$$

⁶ Il nome deriva dall’abitudine degli attuari di scrivere la somma delle frequenze osservate sui margini delle tabelle assicurative.

Tanto per essere sicuri, facciamo la prova calcolando la probabilità che dato il mal di denti *non* ci sia carie:

$$\begin{aligned} P(\neg\text{carie} \mid \text{maldimenti}) &= \frac{P(\neg\text{carie} \wedge \text{maldimenti})}{P(\text{maldimenti})} \\ &= \frac{0,0168 + 0,064}{0,108 + 0,012 + 0,016 + 0,064} = 0,4. \end{aligned}$$

La somma dei due valori è, correttamente 1,0. Notate che il termine $P(\text{maldimenti})$ sta al denominatore in entrambi i calcoli precedenti. Se la variabile *Carie* avesse più di due valori, tale termine sarebbe al denominatore di tutti. In effetti, il termine può essere considerato una costante di **normalizzazione** per la distribuzione $\mathbf{P}(\text{Carie}|\text{maldimenti})$, che assicura che la sua somma valga 1. In tutti i capitoli che trattano la probabilità useremo la lettera α per indicare simili costanti. Con questa notazione, possiamo scrivere le due precedenti equazioni come una sola:

$$\begin{aligned} \mathbf{P}(\text{Carie}|\text{maldimenti}) &= \alpha \mathbf{P}(\text{Carie}, \text{maldimenti}) \\ &= \alpha [\mathbf{P}(\text{Carie}, \text{maldimenti}, \text{prende}) + \mathbf{P}(\text{Carie}, \text{maldimenti}, \neg\text{prende})] \\ &= \alpha [\langle 0,108, 0,016 \rangle + \langle 0,012, 0,064 \rangle] = \alpha \langle 0,12, 0,08 \rangle = \langle 0,6, 0,4 \rangle. \end{aligned}$$

In altre parole, possiamo calcolare $\mathbf{P}(\text{Carie}|\text{maldimenti})$ anche se non conosciamo il valore di $P(\text{maldimenti})$! Dimentichiamo per un momento il fattore $1 / P(\text{maldimenti})$ e sommiamo i valori per *carie* e $\neg\text{carie}$, ottenendo 0,12 e 0,08. Questi valori sono in proporzioni relative corrette, ma la loro somma non è 1, perciò li normalizziamo dividendoli entrambi per 0,12 + 0,08, ottenendo le vere probabilità: 0,6 e 0,4. La normalizzazione si rivela un'utile scoriaia in molti calcoli relativi alle probabilità, sia per facilitarli, sia per consentirci di procedere quando non sono disponibili alcune valutazioni probabilistiche (come $P(\text{maldimenti})$).

Da questo esempio possiamo ricavare una procedura di inferenza generale. Iniziamo con il caso in cui l'interrogazione riguarda una sola variabile, X (nell'esempio, *Carie*). Sia **E** la lista di variabili dell'evidenza (nell'esempio, solo *MalDiDenti*), sia **e** la lista dei valori osservati per esse e sia **Y** l'insieme delle restanti variabili non osservate (nell'esempio, solo *Prende*). La query è $\mathbf{P}(X|\mathbf{e})$ e può essere valutata come

$$\mathbf{P}(X|\mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}), \quad (12.9)$$

dove la somma spazia su tutte le possibili **y** (cioè, tutte le possibili combinazioni di valori delle variabili non osservate **Y**). Notate che, prese insieme, le variabili X , **E** e **Y** costituiscono l'insieme completo di variabili del dominio, cosicché $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$ è semplicemente un sottoinsieme delle probabilità della distribuzione congiunta completa.

Data la distribuzione congiunta completa su cui lavorare, l'Equazione (12.9) può rispondere a interrogazioni probabilistiche su variabili discrete. L'algoritmo tuttavia non scala molto bene: se il dominio è descritto da n variabili booleane, le dimensioni della tabella di input devono essere $O(2^n)$, così come il tempo richiesto per elaborarla. In un problema realistico potremmo avere $n = 100$, per cui $O(2^n)$ sarebbe praticamente impossibile da elaborare – avremmo una tabella con $2^{100} \approx 10^{30}$ elementi. Non è solo un problema di memoria e di tempo di calcolo: il problema vero è che ognuna delle 10^{30} probabilità deve essere stimata separatamente a partire da esempi, per cui il numero di esempi richiesti sarebbe astronomico.

Per queste ragioni, la distribuzione congiunta completa in forma tabellare è uno strumento utilizzabile soltanto raramente per costruire sistemi di ragionamento nella pratica. Deve essere piuttosto considerata il fondamento teorico da cui partire per sviluppare approcci più efficaci, analogamente alle tabelle di verità che hanno fornito fondamenta teoriche per algoritmi più pratici come DPLL nel Capitolo 7. Nel resto di questo capitolo introdurremo alcune delle idee di base necessarie per lo sviluppo dei sistemi realistici del Capitolo 13.

12.4 Indipendenza

Espandiamo la distribuzione congiunta completa della Figura 12.3 aggiungendo una quarta variabile, *CondizioniAtmosferiche*. A questo punto la distribuzione diventa $\mathbf{P}(MalDiDenti, Prende, Carie, CondizioniAtmosferiche)$, che ha $2 \times 2 \times 2 \times 4 = 32$ elementi. Si può pensare di rappresentarla con quattro diverse “edizioni” della tabella mostrata nella Figura 12.3, una per ogni tipo di condizione meteo. È naturale chiedersi quale sia la relazione di queste edizioni tra loro e con la tabella originale a tre variabili, nonché quale relazione ci sia tra $P(maldidenti, prende, carie, CondizioniAtmosferiche = coperto)$ e $P(maldidenti, prende, carie)$. Possiamo usare la regola del prodotto:

$$\begin{aligned} P(maldidenti, prende, carie, coperto) \\ = P(coperto|maldidenti, prende, carie) P(maldidenti, prende, carie). \end{aligned}$$

Ora, a meno che non si parli di divinità vendicative, è difficile immaginare che eventuali problemi dentari possano influenzare le condizioni meteorologiche. E inoltre, almeno per i dentisti che operano in locali al chiuso, si può affermare che le condizioni meteo non influenzino le variabili legate ai denti. Sembra quindi ragionevole affermare quanto segue:

$$P(coperto|maldidenti, prende, carie) = P(coperto). \quad (12.10)$$

Da questo possiamo dedurre:

$$P(maldidenti, prende, carie, coperto) = P(coperto)P(maldidenti, prende, carie).$$

Si può scrivere un’equazione simile per ogni elemento di $\mathbf{P}(MalDiDenti, Prende, Carie, CondizioniAtmosferiche)$. In effetti, possiamo scrivere l’equazione generale:

$$\begin{aligned} \mathbf{P}(MalDiDenti, Prende, Carie, CondizioniAtmosferiche) \\ = \mathbf{P}(MalDiDenti, Prende, Carie)\mathbf{P}(CondizioniAtmosferiche). \end{aligned}$$

In definitiva, la tabella di 32 elementi per le quattro variabili può essere costruita partendo da una tabella di 8 e una di 4 elementi. Questa scomposizione è rappresentata schematicamente nella Figura 12.4(a).

indipendenza

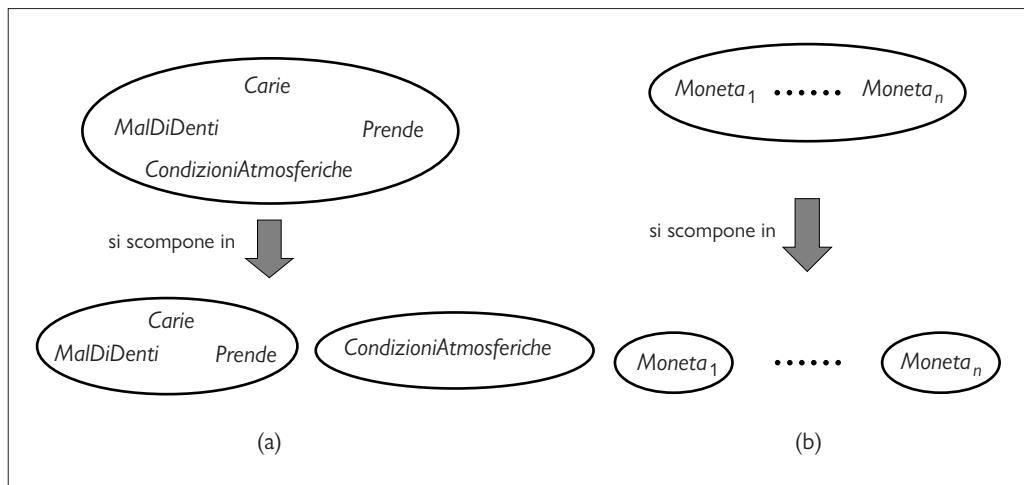
La proprietà che abbiamo sfruttato scrivendo l’Equazione (12.10) prende il nome di **indipendenza** (alternativamente, **indipendenza marginale** o **indipendenza assoluta**). In particolare, in questo caso le condizioni meteo sono indipendenti dai nostri problemi dentari. L’indipendenza tra le proposizioni a e b può essere scritta come

$$P(a|b) = P(a) \text{ oppure } P(b|a) = P(b) \text{ oppure } P(a \wedge b) = P(a)P(b). \quad (12.11)$$

Figura 12.4

Due esempi di fattorizzazione di una grande distribuzione congiunta in distribuzioni più piccole, sfruttando l’indipendenza assoluta.

- (a) Le condizioni meteo e i problemi dentari sono indipendenti.
- (b) I lanci di una moneta sono indipendenti tra loro.



Queste forme sono tutte equivalenti (cfr. Esercizio 12.INDI). L'indipendenza tra le variabili X e Y può essere scritta come segue (ancora una volta, le forme sono tutte equivalenti):

$$\mathbf{P}(X|Y) = \mathbf{P}(X) \quad \text{oppure} \quad \mathbf{P}(Y|X) = \mathbf{P}(Y) \quad \text{oppure} \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y).$$

Normalmente le asserzioni che riguardano l'indipendenza sono basate sulla conoscenza del dominio. Come mostra l'esempio dei problemi dentali e delle condizioni meteo, tali asserzioni possono ridurre sensibilmente la quantità di informazione necessaria per specificare la distribuzione congiunta completa. Se l'insieme delle variabili può essere diviso in sottoinsiemi indipendenti, la distribuzione congiunta completa può essere *fattorizzata* in distribuzioni congiunte separate sui sottoinsiemi. Per esempio, la distribuzione congiunta completa sull'esito di n lanci di moneta indipendenti, $\mathbf{P}(C_1, \dots, C_n)$ ha 2^n elementi, ma può essere rappresentata dal prodotto di n distribuzioni a variabile singola $\mathbf{P}(C_i)$. In senso pratico l'indipendenza di odontoiatria e meteorologia è una cosa buona, perché altrimenti per esercitare il mestiere del dentista occorrerebbe conoscere profondamente quello del meteorologo e viceversa.

In definitiva, quando sono disponibili, le asserzioni di indipendenza possono aiutare a ridurre drasticamente la dimensione della rappresentazione del dominio e quindi la complessità del problema inferenziale. Sfortunatamente è piuttosto raro riuscire a separare netta mente interi insiemi di variabili attraverso l'indipendenza. Ogniqualvolta c'è una connessione tra due variabili, per quanto tenue e indiretta, l'indipendenza non sarà verificata. Inoltre, anche gli insiemi indipendenti possono risultare piuttosto estesi – l'odontoiatria potrebbe coinvolgere dozzine di patologie e centinaia di sintomi, tutti collegati. Per gestire problemi simili, occorrono metodi più sottili del semplice concetto di indipendenza.

12.5 La regola di Bayes e il suo utilizzo

Nel Paragrafo 12.2.1 abbiamo definito la **regola del prodotto** (Equazione (12.4)), che in effetti può essere scritta in due forme:

$$P(a \wedge b) = P(a|b)P(b) \quad \text{e} \quad P(a \wedge b) = P(b|a)P(a).$$

Scrivendo l'uguaglianza tra le due parti destre e dividendo per $P(a)$ otteniamo:

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)}. \tag{12.12}$$

Questa semplice equazione è nota come **regola di Bayes** (o anche legge o teorema di Bayes) ed è alla base di tutti i sistemi moderni di inferenza probabilistica.

regola di Bayes

Il caso più generale della regola di Bayes con variabili a più valori può essere scritto, ricorrendo alla notazione **P**, come segue:

$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(X|Y)\mathbf{P}(Y)}{\mathbf{P}(X)}.$$

Come prima, questa formula dev'essere interpretata come un intero insieme di equazioni, ognuna riferita a un valore specifico delle variabili. Ci capiterà anche di usare una versione condizionata su una evidenza **e**:

$$\mathbf{P}(Y|X, \mathbf{e}) = \frac{\mathbf{P}(X|Y, \mathbf{e})\mathbf{P}(Y|\mathbf{e})}{\mathbf{P}(X|\mathbf{e})}. \tag{12.13}$$

12.5.1 Applicare la regola di Bayes: il caso semplice

Di primo acchito, la regola di Bayes non sembra poi tanto utile. Ci consente di calcolare il singolo termine $P(b|a)$ usando i tre termini $P(a|b)$, $P(b)$ e $P(a)$. Sembra di fare due passi indietro, tuttavia la regola di Bayes è utile nella pratica perché ci sono molti casi in cui abbiamo buone stime di questi tre numeri e un’effettiva necessità di calcolare il quarto. Spesso percepiamo come evidenza l’*effetto* di una *causa* ignota che vorremmo determinare. In tal caso, la regola di Bayes diventa:

$$P(\text{causa} | \text{effetto}) = \frac{P(\text{effetto} | \text{causa}) P(\text{causa})}{P(\text{effetto})}$$

causale
diagnostica

La probabilità condizionata $P(\text{effetto} | \text{causa})$ quantifica la relazione nella direzione **causale**, mentre $P(\text{causa} | \text{effetto})$ descrive la direzione **diagnostica**. In un’attività come la diagnosi medica, spesso abbiamo a disposizione le probabilità condizionate delle relazioni causali. Il medico conosce $P(\text{sintomi} | \text{malattia})$ e vuole ricavare una diagnosi, $P(\text{malattia} | \text{sintomi})$.

Per esempio, un medico sa che la malattia meningite causa un torcicollo al paziente, diciamo, nel 70% dei casi. Conosce anche alcuni fatti non condizionati: la probabilità a priori che un qualsiasi paziente abbia la meningite è pari a 1/50.000, e quella che abbia il torcicollo è l’1%. Indicando con s la proposizione che il paziente ha il torcicollo e con m la proposizione che il paziente ha la meningite, abbiamo

$$\begin{aligned} P(s | m) &= 0,7 \\ P(m) &= 1/50000 \\ P(s) &= 0,01 \\ P(m | s) &= \frac{P(s | m) P(m)}{P(s)} = \frac{0,7 \times 1/50000}{0,01} = 0,0014. \end{aligned} \tag{12.14}$$

Quindi ci aspettiamo che solo lo 0,14% dei pazienti con il torcicollo abbia la meningite. Notate che, anche se il torcicollo è un sintomo che si verifica abbastanza frequentemente nei casi di meningite (con probabilità 0,7), la probabilità di meningite nei pazienti con torcicollo rimane comunque piccola. Questo è dovuto al fatto che la probabilità a priori del torcicollo (da qualsiasi causa) è molto superiore alla probabilità a priori della meningite.

Nel Paragrafo 12.3 abbiamo illustrato un processo mediante cui si può evitare di stimare la probabilità dell’evidenza (qui $P(s)$) calcolando invece una probabilità a posteriori per ogni valore della variabile di query (qui, m e $\neg m$) e normalizzando i risultati. Lo stesso processo può essere applicato alla regola di Bayes: risulta

$$\mathbf{P}(M|s) = \alpha \langle P(s|m)P(m), P(s|\neg m)P(\neg m) \rangle.$$

Per utilizzare quest’approccio dobbiamo quindi stimare $P(s|\neg m)$ invece di $P(s)$. Questo non è necessariamente banale: talvolta è più facile, altre volte più difficile. La forma generale della regola di Bayes con la normalizzazione è

$$\mathbf{P}(Y|X) = \alpha \mathbf{P}(X|Y)\mathbf{P}(Y), \tag{12.11}$$

dove α è la costante di normalizzazione necessaria affinché la somma degli elementi di $\mathbf{P}(Y|X)$ sia pari a 1.

Una domanda che viene subito in mente circa la regola di Bayes è: perché dovrebbe essere disponibile la probabilità condizionata in una direzione, ma non nell’altra? Per quanto riguarda la meningite, può darsi che il dottore sappia che il torcicollo implica la meningite in un caso su 5000; in altre parole il dottore potrebbe possedere informazione quantitativa nella direzione **diagnostica** che risale dai sintomi alle cause. Un dottore così edotto non ha bisogno di ricorrere alla regola di Bayes.



Sfortunatamente, la conoscenza diagnostica è spesso più fragile di quella causale. Se si dovesse verificare un'improvvisa epidemia di meningite, la sua probabilità non condizionata $P(m)$ salirebbe. Il dottore che ha derivato la probabilità diagnostica $P(m|s)$ direttamente dall'osservazione statistica dei pazienti prima dell'epidemia non saprà come aggiornare il valore, ma quello che calcola $P(m|s)$ dagli altri tre valori vedrà che $P(m|s)$ cresce in proporzione a $P(m)$. Inoltre, ciò che è più importante, l'informazione causale $P(s|m)$ non è influenzata dall'epidemia, dato che riflette semplicemente il funzionamento della meningite. Utilizzare questo tipo di conoscenza causale diretta, o basata sul modello, fornisce la robustezza indispensabile affinché i sistemi probabilistici possano operare nel mondo reale.

12.5.2 Usare la regola di Bayes: combinazione di evidenze

Abbiamo visto che la regola di Bayes può essere utile per rispondere a interrogazioni probabilistiche condizionate su un singolo elemento di evidenza: nell'esempio proposto, il torcicollo. In particolare, abbiamo notato che spesso l'informazione probabilistica è disponibile nella forma $P(\text{effetto}|\text{causa})$. Che cosa succede quando abbiamo due o più elementi di evidenza? Per esempio, cosa può concludere un dentista quando il suo temibile strumento d'acciaio "prende" nel dente dolorante di un paziente? Se conosciamo la distribuzione congiunta completa (Figura 12.3) possiamo leggere semplicemente la risposta:

$$\mathbf{P}(\text{Carie}|\text{maldimenti} \wedge \text{prende}) = \alpha \langle 0,108, 0,016 \rangle \approx \langle 0,871, 0,129 \rangle.$$

Tuttavia, sappiamo che un simile approccio non scala al crescere del numero delle variabili. Possiamo provare a riformulare il problema usando la regola di Bayes:

$$\mathbf{P}(\text{Carie}|\text{maldimenti} \wedge \text{prende}) = \alpha \mathbf{P}(\text{maldimenti} \wedge \text{prende}|\text{Carie})\mathbf{P}(\text{Carie}). \quad (12.16)$$

Affinché questa riformulazione possa funzionare, occorre conoscere le probabilità condizionate della congiunzione $\text{maldimenti} \wedge \text{prende}$ per ogni valore di Carie . Ancora una volta, questo potrebbe essere fattibile con una evidenza costituita da due variabili, ma non scala. Infatti, con n variabili di evidenza (radiografie, dieta, igiene orale etc.) risulteranno $O(2^n)$ combinazioni possibili di valori osservati di cui dovremmo conoscere le probabilità condizionate: a questo punto conviene usare la distribuzione congiunta completa.

Per fare progressi, dovremo trovare asserzioni aggiuntive riguardanti il dominio che ci permettano di semplificare le espressioni. La nozione di **indipendenza** presentata nel Paragrafo 12.4 è un buon inizio, ma dev'essere raffinata. Sarebbe bello se *MalDiDenti* e *Prende* fossero indipendenti, ma non lo sono: se lo strumento si incasca nel dente, è probabile che quest'ultimo abbia una carie e che questa sia la causa del dolore. Le variabili *sono* indipendenti, però, *data la presenza o assenza della carie*. Ognuna di esse è causata dalla carie, ma nessuna ha un effetto diretto sull'altra: il dolore dipende principalmente dallo stato dei nervi all'interno del dente, mentre l'accuratezza dello strumento dipende dalla bravura del dentista, che non ha alcun rapporto con il mal di denti.⁷ In termini matematici questa proprietà si scrive:

$$\mathbf{P}(\text{maldimenti} \wedge \text{prende}|\text{Carie}) = \mathbf{P}(\text{maldimenti}|\text{Carie})\mathbf{P}(\text{prende}|\text{Carie}). \quad (12.17)$$

Questa equazione esprime l'**indipendenza condizionale** di *maldimenti* e *prende* data *Carie*. Possiamo sostituirla nell'Equazione (12.16) per ottenere la probabilità della carie:

indipendenza condizionale

$$\mathbf{P}(\text{Carie}|\text{maldimenti} \wedge \text{prende}) = \alpha \mathbf{P}(\text{maldimenti}|\text{Carie})\mathbf{P}(\text{prende}|\text{Carie}) \mathbf{P}(\text{Carie}). \quad (12.18)$$

⁷ Ipotizziamo che il paziente e il dentista siano individui distinti.

Ora i requisiti, per quanto riguarda le informazioni necessarie, sono gli stessi dell’inferenza che utilizza ogni evidenza separatamente: la probabilità a priori $\mathbf{P}(Carie)$ della variabile della query e quella condizionata di ogni effetto, data la sua causa.

La definizione generale di indipendenza condizionale di due variabili X e Y , data una terza variabile Z , è

$$\mathbf{P}(X, Y|Z) = \mathbf{P}(X|Z)\mathbf{P}(Y|Z).$$

Nel dominio dentistico, per esempio, sembra ragionevole asserire l’indipendenza condizionale delle variabili *MalDiDenti* e *Prende*, data *Carie*:

$$\mathbf{P}(MalDiDenti, Prende|Carie) = \mathbf{P}(MalDiDenti|Carie)\mathbf{P}(Prende|Carie). \quad (12.19)$$

Notate che quest’asserzione è più forte dell’Equazione (12.17), che afferma l’indipendenza solo per valori specifici di *MalDiDenti* e *Prende*. Come nel caso dell’indipendenza assoluta dell’Equazione (12.11), si possono usare anche le forme alternative

$$\mathbf{P}(X|Y, Z) = \mathbf{P}(X|Z) \quad \text{e} \quad \mathbf{P}(Y|X, Z) = \mathbf{P}(Y|Z)$$

(cfr. Esercizio 12.PXYZ). Il Paragrafo 12.4 ha mostrato che l’indipendenza assoluta permette di scomporre la distribuzione congiunta completa in parti molto più piccole. Lo stesso si può dire dell’indipendenza condizionata. Per esempio, data l’asserzione dell’Equazione (12.19), possiamo derivare una scomposizione come segue:

$$\begin{aligned} \mathbf{P}(MalDiDenti, Prende, Carie) \\ &= \mathbf{P}(MalDiDenti, Prende|Carie)\mathbf{P}(Carie) \quad (\text{regola del prodotto}) \\ &= \mathbf{P}(MalDiDenti|Carie)\mathbf{P}(Prende|Carie)\mathbf{P}(Carie) \quad (\text{applicando (12.19)}). \end{aligned}$$

È facile verificare che questa equazione vale realmente osservando la Figura 12.3.

In questo modo, la grande tabella originale è scomposta in tre tabelle più piccole. Quella originale aveva sette numeri indipendenti (la tabella ha $2^3 = 8$ elementi, ma la loro somma deve essere 1, perciò 7 sono indipendenti). Le tabelle più piccole contengono un totale di $2 + 2 + 1 = 5$ numeri indipendenti (per una distribuzione di probabilità condizionata come $\mathbf{P}(MalDiDenti|Carie)$ ci sono due righe di due numeri, e la somma degli elementi di ogni riga è 1, perciò sono due numeri indipendenti; per una distribuzione di probabilità a priori come $\mathbf{P}(Carie)$ c’è un solo numero indipendente). Passare da 7 a 5 potrebbe non sembrare un gran progresso, ma si possono ottenere risultati molto più significativi con un maggior numero di sintomi.

In generale, per n sintomi tutti condizionalmente indipendenti data *Carie*, la dimensione della rappresentazione cresce con $O(n)$ invece di $O(2^n)$. Questo significa che *le asserzioni di indipendenza condizionale possono permettere ai sistemi probabilistici di scalare verso l’alto; inoltre sono molto più facili da ottenere di quelle che riguardano l’indipendenza assoluta*. Concettualmente, *Carie separa MalDiDenti e Prende* perché è la causa diretta di entrambe. La scomposizione attraverso l’indipendenza condizionale di grandi domini probabilistici in sottoinsiemi debolmente collegati è uno degli sviluppi più importanti nella storia recente dell’IA.



12.6 Modelli di Bayes ingenui

L’esempio dentistico illustra uno schema che si verifica di frequente, in cui una singola causa influenza direttamente più effetti, tutti condizionalmente indipendenti data la causa. La distribuzione congiunta completa può essere in tal caso scritta:

$$\mathbf{P}(Causa, Effetto_1, \dots, Effetto_n) = \mathbf{P}(Causa) \prod_i \mathbf{P}(Effetto_i | Causa). \quad (12.20)$$

Una simile distribuzione di probabilità prende il nome di **modello di Bayes ingenuo** (*naive Bayes*): la sua “ingenuità” deriva dal fatto che viene spesso usata (come ipotesi semplificativa) anche in casi in cui le variabili “di effetto” *non* sono strettamente indipendenti data la causa. Il modello ingenuo di Bayes viene talvolta chiamato **classificatore bayesiano**, un uso alquanto improprio che ha spinto i veri bayesiani a ribattezzarlo **modello di Bayes idiota**. Nella pratica, spesso i sistemi di Bayes ingenui funzionano sorprendentemente bene, anche quando l’assunzione dell’indipendenza condizionale non è strettamente verificata.

**modello di Bayes
ingenuo**

Per usare un modello di Bayes ingenuo possiamo applicare l’Equazione (12.20) per ottenere la probabilità della causa dati alcuni effetti osservati. Chiamiamo $\mathbf{E} = \mathbf{e}$ gli effetti osservati, mentre le rimanenti variabili di effetto \mathbf{Y} sono non osservate. Allora si può applicare il metodo standard per inferire dalla distribuzione congiunta (Equazione (12.9)):

$$\mathbf{P}(Causa | \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(Causa, \mathbf{e}, \mathbf{y}).$$

Dall’Equazione (12.20) otteniamo quindi

$$\begin{aligned} \mathbf{P}(Causa | \mathbf{e}) &= \alpha \sum_{\mathbf{y}} \mathbf{P}(Causa) \mathbf{P}(\mathbf{y} | Causa) \left(\prod_j \mathbf{P}(e_j | Causa) \right) \\ &= \alpha \mathbf{P}(Causa) \left(\prod_j \mathbf{P}(e_j | Causa) \right) \sum_{\mathbf{y}} \mathbf{P}(\mathbf{y} | Causa) \\ &= \alpha \mathbf{P}(Causa) \prod_j \mathbf{P}(e_j | Causa) \end{aligned} \quad (12.21)$$

dove l’ultima riga segue dal fatto che la somma su \mathbf{y} è 1. Possiamo reinterpretare questa equazione nei seguenti termini: per ogni possibile causa, moltiplichiamo la probabilità a priori della causa per il prodotto delle probabilità condizionate degli effetti osservati data la causa, poi normalizziamo il risultato. Il tempo di esecuzione di questo calcolo è lineare nel numero di effetti osservati e non dipende dal numero di effetti non osservati (che può essere molto alto in domini come per esempio la medicina). Nel Capitolo 13 vedremo che questo è un fenomeno comune nell’inferenza probabilistica: le variabili delle evidenze i cui valori sono non osservati solitamente “scompaiono” del tutto dal calcolo.

12.6.1 Classificazione di testi con un modello Bayes ingenuo

Vediamo ora come utilizzare un modello di Bayes ingenuo per l’attività di **classificazione di testi**: dato un testo, decidere a quale insieme predefinito di classi o categorie appartenga. In questo caso la “causa” è la variabile *Categoria* e le variabili “effetto” sono la presenza o l’assenza di determinate parole chiave, $ContieneParola_i$. Consideriamo due frasi di esempio tratte da articoli di quotidiani:

**classificazione
di testi**

1. Lunedì i prezzi delle azioni sono saliti molto e i principali indici hanno guadagnato l’1% per l’ottimismo sui risultati aziendali del primo trimestre.
2. Forti piogge hanno colpito incessantemente gran parte della costa est lunedì, avvisi di possibili alluvioni sono stati diramati a New York City e in altre località.

Il compito è classificare ciascuna frase in una *Categoria* – in questo caso le principali sezioni del quotidiano: *notizie*, *sport*, *economia*, *meteo* o *intrattenimento*. Il modello di Bayes ingenuo è costituito dalle probabilità a priori $\mathbf{P}(Categoria)$ e dalle probabilità condizionate $\mathbf{P}(ContieneParola_i | Categoria)$.

Per ogni categoria c , $P(Categoria = c)$ è stimata come la quota di tutti i documenti precedentemente consultati che appartengono alla categoria c . Per esempio, se il 9% degli articoli tratta delle condizioni meteo, poniamo $P(Categoria = meteo) = 0,09$. Analogamente, $\mathbf{P}(ContieneParola_i | Categoria)$ è stimata come la quota dei documenti di ciascuna categoria che

contiene la parola i ; se il 37% degli articoli di economia contiene la parola 6, “azioni”, allora $P(\text{ContieneParola}_6 = \text{true} | \text{Categoria} = \text{economia})$ è posta a 0,37.⁸

Per assegnare la categoria a un nuovo documento, controlliamo quali parole chiave appaiono in esso e poi applichiamo l’Equazione (12.21) per ottenere la distribuzione di probabilità a posteriori sulle categorie. Se dobbiamo predire una sola categoria, consideriamo quella con la probabilità a posteriori più alta. Notate che per questa attività vengono osservate tutte le variabili effetto, poiché possiamo sempre affermare se una data parola appare nel documento o meno.

Il modello di Bayes ingenuo assume che le parole siano presenti in modo indipendente all’interno dei documenti, con frequenze determinate dalla categoria del documento. Questo assunto di indipendenza è chiaramente violato nella pratica. Per esempio, la frase “primo trimestre” appare negli articoli di economia più frequentemente di quanto suggerirebbe una semplice moltiplicazione delle probabilità di “primo” e “trimestre”. La violazione dell’assunto di indipendenza significa solitamente che le probabilità a posteriori saranno molto più vicine a 1 o 0 rispetto al previsto; in altre parole, il modello è eccessivamente fiducioso nelle sue previsioni. D’altra parte, anche tenendo conto di questi errori, l’*ordinamento* delle possibili categorie è spesso accurata.

I modelli di Bayes ingenui sono spesso usati per compiti di analisi linguistica, gestione di documenti, filtri di spam e altre attività di classificazione. Per attività come la diagnosi medica, in cui i valori effettivi delle probabilità a posteriori hanno grande importanza – per esempio nel decidere se eseguire o meno un’appendicectomia – solitamente si preferisce utilizzare modelli più avanzati come quelli descritti nel Capitolo 13.

12.7 Il mondo del wumpus rivisitato

È possibile combinare le idee presentate in questo capitolo per risolvere problemi di ragionamento probabilistico nel mondo del wumpus (cfr. Capitolo 7 per una sua descrizione completa). In questo ambiente l’incertezza sorge dal fatto che i sensori dell’agente forniscono solo un’informazione parziale sul mondo. Per esempio, la Figura 12.5 illustra una situazione in cui ognuna delle tre stanze non visitate ma raggiungibili – [1,3], [2,2] e [3,1] – potrebbe contenere un pozzo. L’inferenza logica pura non può concludere quale stanza avrà più probabilità di essere sicura, ragion per cui un agente logico deve scegliere casualmente. Come vedremo, un agente probabilistico può comportarsi molto meglio.

Il nostro scopo è calcolare la probabilità che ognuna delle tre stanze contenga un pozzo (ignorando per semplicità la posizione del wumpus e dell’oro). Le proprietà rilevanti del mondo sono che (1) un pozzo causa uno spostamento d’aria percepibile in tutte le stanze adiacenti, e (2) ogni stanza tranne [1,1] ha una probabilità pari a 0,2 di contenere un pozzo. Per prima cosa identifichiamo l’insieme di variabili casuali che ci occorrono:

- come nel caso logico proposizionale, una variabile booleana P_{ij} per ogni stanza, che è vera se e solo se $[i, j]$ contiene effettivamente un pozzo;
- variabili booleane B_{ij} , vere se e solo se la stanza $[i, j]$ è ventosa. Includeremo solo le variabili relative alle stanze che abbiamo visitato: in questo caso [1,1], [1,2] e [2,1].

⁸ Occorre prestare attenzione a non assegnare probabilità nulla a parole che non sono ancora state rilevate in una data categoria di documenti, poiché il valore zero eliminerebbe ogni altra evidenza nell’Equazione (12.21). Il fatto che non abbiate ancora visto una parola non significa che non la vedrete mai. Occorre invece riservare una piccola porzione della distribuzione di probabilità per rappresentare parole “precedentemente non viste”. Cfr. il Capitolo 20 del Volume 2 per un approfondimento di questo aspetto in generale e il Paragrafo 23.1.4 nel Capitolo 23 del Volume 2 per il caso particolare dei modelli di parole.

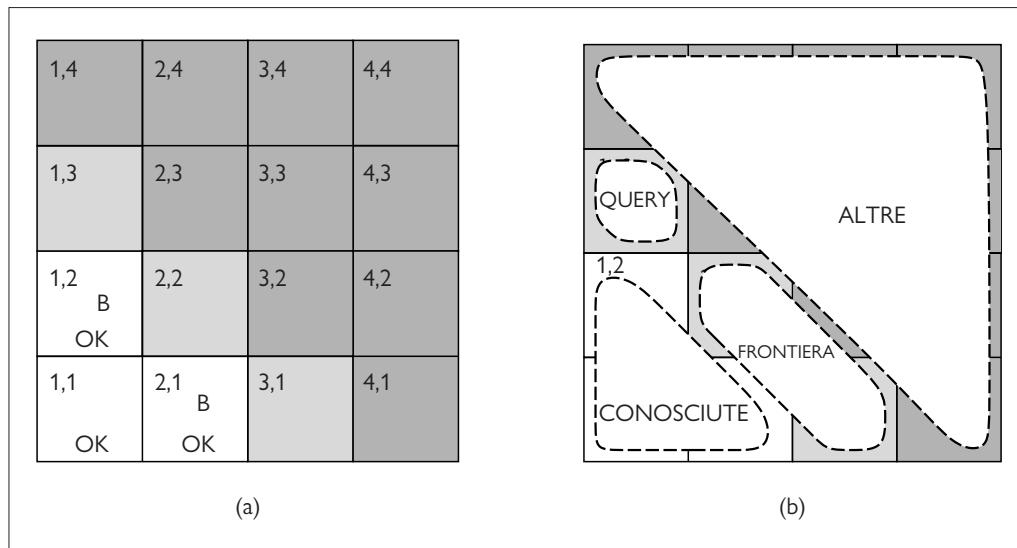


Figura 12.5
 (a) Dopo aver rilevato una brezza sia in [1,2] che in [2,1], l'agente è bloccato: non c'è più alcuna stanza sicura da esplorare.
 (b) Suddivisione delle stanze in *Conosciute*, *Frontiera* e *Altre*, per un'interrogazione riguardante [1,3].

Il passo successivo è specificare la distribuzione congiunta completa, $\mathbf{P}(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$. Applicando la regola del prodotto otteniamo

$$\begin{aligned} \mathbf{P}(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) &= \\ \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} | P_{1,1}, \dots, P_{4,4}) \mathbf{P}(P_{1,1}, \dots, P_{4,4}). \end{aligned}$$

Questa scomposizione rende facile vedere quali devono essere i valori della distribuzione congiunta. Il primo termine è la distribuzione di probabilità condizionata della brezza, data la configurazione dei pozzi; i suoi valori sono 1 se tutte le stanze con brezza sono adiacenti ai pozzi e 0 in caso contrario. Il secondo termine è la probabilità a priori della presenza di un pozzo. Ogni stanza contiene un pozzo con probabilità 0,2, indipendentemente dalle altre posizioni; per cui

$$\mathbf{P}(P_{1,1}, \dots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}). \quad (12.22)$$

Per una particolare configurazione con esattamente n pozzi, la probabilità è $0,2^n \times 0,8^{16-n}$.

Nella situazione della Figura 12.5(a), l'evidenza consiste nella rilevazione di uno spostamento d'aria (o della sua assenza) in ogni stanza visitata, unita al fatto che nessuna di tali stanze contiene un pozzo. Abbreviamo questi fatti scrivendo $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ e $conosciute = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$. Quello che ci interessa è rispondere a query come $\mathbf{P}(P_{1,3} | conosciute, b)$: che probabilità c'è che [1,3] contenga un pozzo, date le percezioni rilevate sin qui?

Per rispondere a quest'interrogazione possiamo seguire l'approccio standard suggerito dall'Equazione (12.9), sommando tutti gli elementi rilevanti della distribuzione congiunta completa. Definiamo *Sconosciute* la variabile composta da tutte le variabili $P_{i,j}$ non comprese in *Conosciute* tranne la stanza della query, [1,3]. Per l'Equazione (12.9) risulta

$$\mathbf{P}(P_{1,3} | conosciute, b) = \alpha \prod_{sconosciute} \mathbf{P}(P_{1,3}, conosciute, b, sconosciute). \quad (12.23)$$

Le probabilità congiunte sono già state tutte specificate, per cui dovremmo aver finito... se non ci interessa la complessità computazionale. Le stanze sconosciute sono 12; la somatoria conterrà quindi $2^{12} = 4096$ termini. In generale, la somma crescerà esponenzialmente con il numero di stanze.

Ci si potrebbe chiedere se le altre stanze non dovrebbero essere irrilevanti. Come può il contenuto di [4,4] avere effetto sulla presenza o meno di un pozzo in [1,3]? Invero, questa intuizione è più o meno corretta, ma occorre renderla più precisa. Ciò che intendiamo in realtà è che, se conosciamo i valori di tutte le variabili pozzo adiacenti alle stanze che ci interessano, allora i pozzi (o la loro assenza) in altre stanze più distanti non potrebbero avere altro effetto sulla nostra credenza.

Chiamiamo *Frontiera* le variabili (tranne quella della query) adiacenti a stanze già visitate, in questo caso solo [2,2] e [3,1]. Inoltre, chiamiamo *Altre* tutte le altre variabili pozzo relative a stanze sconosciute; in questo caso ce ne sono 10, come si vede nella Figura 12.5(b). Con queste definizioni, *Sconosciute* = *Frontiera* ∪ *Altre*.

L’intuizione chiave può essere descritta come segue: le brezze osservate sono *condizionatamente indipendenti* dalle altre variabili, date le variabili per le stanze conosciute, la frontiera e la query. Per mettere a frutto questa intuizione dobbiamo formulare la query in modo che le brezze siano condizionate a tutte le altre variabili, e poi applicare l’indipendenza condizionale:

$$\begin{aligned} & \mathbf{P}(P_{1,3} | \text{conosciute}, b) \\ &= \alpha \sum_{\text{sconosciute}} \mathbf{P}(P_{1,3}, \text{conosciute}, b, \text{sconosciute}) \quad (\text{dall’Equazione (12.23)}) \\ &= \alpha \sum_{\text{sconosciute}} \mathbf{P}(b | P_{1,3}, \text{conosciute}, \text{sconosciute}) \mathbf{P}(P_{1,3}, \text{conosciute}, \text{sconosciute}) \quad (\text{per la regola del prodotto}) \\ &= \alpha \sum_{\text{frontiera altre}} \sum_{\text{frontiera altre}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}, \text{altre}) \mathbf{P}(P_{1,3}, \text{conosciute}, \text{frontiera}, \text{altre}) \\ &= \alpha \sum_{\text{frontiera altre}} \sum_{\text{frontiera altre}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}) \mathbf{P}(P_{1,3}, \text{conosciute}, \text{frontiera}, \text{altre}), \end{aligned}$$

in cui il passo finale utilizza l’indipendenza condizionale: *b* è indipendente da *altre* date *conosciute*. Il primo termine dell’espressione non dipende dalle *Altre* variabili, per cui possiamo spostare la sommatoria all’interno:

$$\begin{aligned} & \mathbf{P}(P_{1,3} | \text{conosciute}, b) \\ &= \alpha \sum_{\text{frontiera}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}) \sum_{\text{altre}} \mathbf{P}(P_{1,3}, \text{conosciute}, \text{frontiera}, \text{altre}). \end{aligned}$$

Come nell’Equazione (12.22), il termine a destra può essere fattorizzato grazie all’indipendenza. Quindi si può procedere a un riordinamento dei termini:

$$\begin{aligned} & \mathbf{P}(P_{1,3} | \text{conosciute}, b) \\ &= \alpha \sum_{\text{frontiera}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}) \sum_{\text{altre}} \mathbf{P}(P_{1,3}) \mathbf{P}(\text{conosciute}) \mathbf{P}(\text{frontiera}) \mathbf{P}(\text{altre}) \\ &= \alpha \mathbf{P}(\text{conosciute}) \mathbf{P}(P_{1,3}) \sum_{\text{frontiera}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}) \mathbf{P}(\text{frontiera}) \sum_{\text{altre}} \mathbf{P}(\text{altre}) \\ &= \alpha' \mathbf{P}(P_{1,3}) \sum_{\text{frontiera}} \mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera}) \mathbf{P}(\text{frontiera}), \end{aligned}$$

in cui l’ultimo passo integra $\mathbf{P}(\text{conosciute})$ nella costante di normalizzazione e sfrutta il fatto che $\sum_{\text{altre}} \mathbf{P}(\text{altre})$ vale 1.

Ora nella sommatoria sulle variabili frontiera $P_{2,2}$ e $P_{3,1}$ ci sono solo quattro termini: l’uso dell’indipendenza (assoluta e condizionale) ha reso possibile ignorare completamente le altre stanze.

Notate che le probabilità in $\mathbf{P}(b | \text{conosciute}, P_{1,3}, \text{frontiera})$ sono 1 quando le osservazioni della brezza sono consistenti con le altre variabili e 0 altrimenti. Così, per ogni valore di $P_{1,3}$ sommiamo sui *modelli logici* delle variabili frontiera che sono consistenti con i fatti noti (confrontate quest’operazione con l’enumerazione di modelli della Figura 7.5 nel Capitolo 7). I

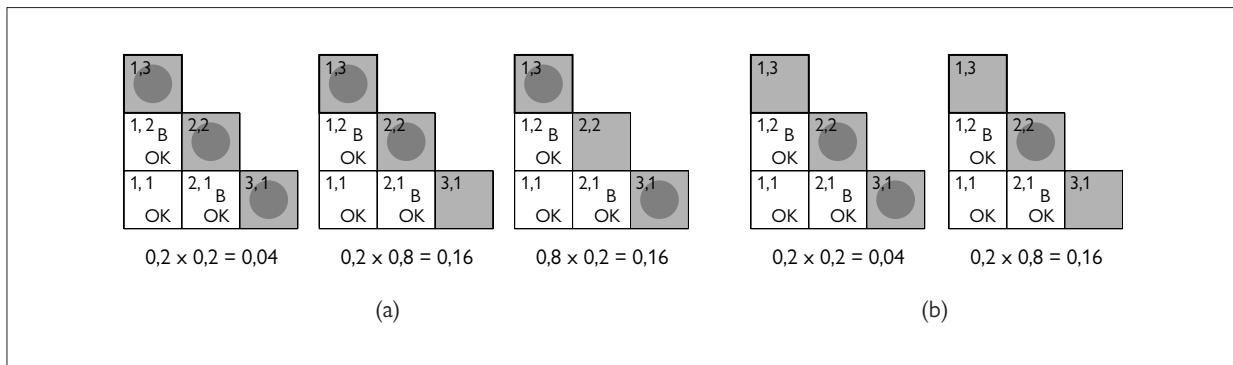


Figura 12.6 Modelli consistenti per le variabili di frontiera $P_{2,2}$ e $P_{3,1}$, con l'indicazione di $P(\text{frontiera})$ per ogni modello: (a) tre modelli con $P_{1,3} = \text{true}$ e la presenza di due o tre pozzi, (b) due modelli con $P_{1,3} = \text{false}$ e la presenza di uno o due pozzi.

modelli e le probabilità a priori $P(\text{frontiera})$ a essi associati sono mostrati nella Figura 12.6. Risulta:

$$\mathbf{P}(P_{1,3} | \text{conosciute}, b) = \alpha' \langle 0,2(0,04 + 0,16 + 0,16), 0,8(0,04 + 0,16) \rangle \approx \langle 0,31, 0,69 \rangle.$$

Questo significa che [1,3] (e [3,1] per simmetria) contengono un pozzo con una probabilità pari al 31% circa. Un calcolo simile, che i lettori potranno svolgere per esercizio, mostra che c'è una probabilità di circa l'86% che [2,2] contenga un pozzo: l'agente vorrà starsene ben lontano. Notate che l'agente logico del Capitolo 7 non sapeva che [2,2] è peggiore delle altre stanze. La logica può dirci che non si sa se vi sia un pozzo in [2,2], ma ci servono le probabilità per sapere quanto è probabile che vi sia un pozzo in quella stanza.

Questo paragrafo ha mostrato che anche problemi apparentemente complicati possono essere formulati precisamente grazie alla teoria della probabilità e risolti con algoritmi semplici. Per ottenere soluzioni *efficienti* si possono sfruttare le relazioni di indipendenza assoluta e condizionale, semplificando così le somme richieste. Queste relazioni corrispondono spesso alla nostra comprensione intuitiva di come si dovrebbe scomporre il problema. Nel prossimo capitolo svilupperemo rappresentazioni formali di tali relazioni e presenteremo algoritmi che operano su di esse per eseguire l'inferenza probabilistica in modo efficiente.

12.8 Riepilogo

In questo capitolo si è suggerito che la teoria della probabilità è un fondamento adatto per ragionare in condizioni di incertezza e si è presentata un'introduzione al suo utilizzo.

- L'incertezza sorge a causa della pigrizia e dell'ignoranza. È una caratteristica inevitabile dei mondi complessi, non deterministici o parzialmente osservabili.
- Le **probabilità** esprimono l'incapacità dell'agente di arrivare a una decisione definitiva sul valore di verità di una formula e servono a riassumere le sue credenze rispetto all'evidenza.
- La **teoria delle decisioni** combina le credenze e i desideri dell'agente, definendo come migliore azione quella che massimizza l'**utilità** attesa.
- Gli enunciati base della probabilità includono le **probabilità a priori** o **non condizionate** e le **probabilità a posteriori** o **condizionate** di proposizioni semplici e complesse.
- Gli assiomi della probabilità vincolano le probabilità di proposizioni in relazione logica tra di loro. Un agente che viola gli assiomi si comporterà irrazionalmente in alcuni casi.
- La **distribuzione di probabilità congiunta completa** specifica la probabilità di ogni assegnamento completo di valori alle variabili casuali. Solitamente è troppo grande per essere

creata o utilizzata in forma esplicita, ma quando è disponibile può essere usata per rispondere a interrogazioni semplicemente sommando gli elementi per i mondi possibili corrispondenti alle proposizioni della query.

- **L’indipendenza assoluta** tra sottoinsiemi di variabili casuali consente di fattorizzare la distribuzione congiunta completa in distribuzioni congiunte più piccole, riducendone notevolmente la complessità.
- **La regola di Bayes** permette di calcolare probabilità sconosciute partendo da probabilità condizionate note, solitamente in direzione causale. In presenza di molti elementi di evidenza, l’applicazione della regola di Bayes avrà gli stessi problemi di scalabilità dell’uso diretto della distribuzione congiunta completa.
- **L’indipendenza condizionale** dovuta a relazioni causalì dirette nel dominio permette di fattorizzare la distribuzione congiunta completa in distribuzioni condizionate più piccole. Il **modello di Bayes ingenuo** assume come ipotesi l’indipendenza condizionale di tutte le variabili effetto data una singola variabile causa e la sua dimensione cresce linearmente con il numero di effetti.
- Un agente del mondo del wumpus può calcolare le probabilità di aspetti del mondo non osservati e avvalersene per prendere decisioni migliori rispetto a un agente puramente logico. L’indipendenza condizionale consente di affrontare questi calcoli.

Note storiche e bibliografiche

La teoria della probabilità fu inventata per analizzare i giochi d’azzardo. Verso l’anno 850 d.C. il matematico indiano Mahaviracarya descrisse come elaborare un insieme di puntate che non potevano perdere (oggi lo chiamiamo *Dutch book* o “libro olandese”). In Europa, le prime analisi sistematiche furono prodotte da Girolamo Cardano intorno al 1565, ma rimasero inedite fino al 1663. A quel punto, la probabilità si era affermata come disciplina matematica grazie a una serie di risultati ottenuti attraverso una famosa corrispondenza tra Blaise Pascal e Pierre Fermat nel 1654. Il primo libro pubblicato sulle probabilità fu *De Rationibus in Ludo Aleae* (Huygens, 1657). La visione di “pigrizia e ignoranza” dell’incertezza fu descritta da John Arbuthnot nella prefazione della sua traduzione di Huygens (Arbuthnot, 1692): “È impossibile che un Dado, con forza e direzione determinata, non cada su un lato determinato, soltanto che non conosco la forza e la direzione che lo fanno cadere su quel determinato lato e perciò la chiamo Fortuna, che non è altro che la volontà dell’arte”.

Il collegamento tra probabilità e ragionamento risale almeno al diciannovesimo secolo: nel 1819 Pierre

Laplace affermò: “La teoria della probabilità non è altro che senso comune ridotto a calcoli”. Nel 1850 James Maxwell disse: “La vera logica di questo mondo si trova nel calcolo delle probabilità, che tiene conto della grandezza della probabilità che sta, o dovrebbe stare, nella mente di un uomo che ragiona”.

Sull’origine e la natura dei valori di probabilità si sono avute infinite discussioni. La posizione **frequentista** sostiene che i valori possano derivare solo dagli *esperimenti*: se esaminiamo 100 persone e verifichiamo che 10 di loro hanno un dente cariato, possiamo affermare che la probabilità di una carie è approssimativamente 0,1. Da questo punto di vista, l’asserzione “la probabilità di una carie è 0,1” significa che 0,1 è la frazione che si osserverebbe al tendere all’infinito del numero degli esperimenti. Dato un campione finito possiamo stimare il valore della frazione e anche calcolare l’accuratezza della nostra stima.

Secondo gli **oggettivisti**, le probabilità sono un vero e proprio aspetto dell’universo – la propensione degli oggetti a comportarsi in un certo modo – e non si limitano a descrivere semplicemente il grado di credenza di un osservatore. Per esempio, il fatto che lancian-

do una moneta bilanciata ci sia una probabilità 0,5 che venga testa è una propensione della moneta stessa. Le misurazioni dei frequentisti, quindi, sarebbero un tentativo di osservare le predisposizioni degli oggetti. La maggior parte dei fisici concorda nel pensare che i fenomeni quantistici siano oggettivamente probabilistici, ma l'incertezza su scala macroscopica – per esempio, nel lancio di una moneta – solitamente scaturisce dall'ignoranza delle condizioni iniziali e non sembra derivare dalle predisposizioni degli oggetti.

La posizione **soggettivista** considera le probabilità un modo di caratterizzare le credenze di un agente, senza assegnare loro alcun significato fisico esterno. La visione soggettiva bayesiana consente di ascrivere le probabilità a priori al ruolo di proposizioni, ma poi insiste su un appropriato aggiornamento bayesiano ove siano fornite evidenze.

Anche una posizione strettamente frequentista richiede un'analisi soggettiva, a causa del problema della **classe di riferimento**: nel tentativo di determinare la probabilità di un *particolare* esperimento, il frequentista deve inserirlo in una classe di riferimento di esperimenti “simili” con frequenze di esito note. Ma qual è la classe giusta? I. J. Good scrisse: “Ogni evento nella vita è unico, e ogni probabilità reale che stimiamo nella pratica è quella di un evento che non si è mai verificato prima” (Good, 1983, p. 27). Per esempio, dato un particolare paziente, un dentista frequentista che voglia conoscere la probabilità di una carie dovrà considerare altri pazienti che presentano importanti caratteristiche in comune – età, sintomi, dieta – e verificare quale proporzione tra essi ha una carie. Se il dentista dovesse considerare tutto ciò che conosce del suo paziente – colore dei capelli, peso preciso al grammo, cognome da nobile della madre e così via – la classe di riferimento sarebbe vuota. Questo è stato uno dei problemi cruciali nella filosofia della scienza.

Pascal utilizzò la probabilità sia nell'interpretazione oggettiva, come una proprietà del mondo basata sulla simmetria o la frequenza relativa, sia in quella soggettiva, basata sui gradi di credenza. In particolare, la sua analisi delle probabilità nei giochi di fortuna si rifaceva al primo approccio, mentre il secondo fu alla base della famosa argomentazione della “scommessa di Pascal” sulla possibile esistenza di Dio. Tuttavia, Pascal non comprese chiaramente la distinzione tra queste due interpretazioni, che fu illustrata chiaramente per la prima volta da James Bernoulli (1654–1705).

Leibniz introdusse la nozione “classica” della probabilità come proporzione di casi enumerati equipro-

babili, usata anche da Bernoulli, sebbene sia stata diffusa ampiamente da Laplace (1816). Questa nozione si pone ambiguumamente a metà tra l'interpretazione come frequenza e quella soggettiva: i casi infatti possono essere considerati ugualmente probabili a causa di una simmetria fisica naturale tra essi, o semplicemente perché non possediamo alcuna informazione per ritenere uno di essi più probabile di un altro. L'uso di quest'ultima considerazione per giustificare l'assegnamento di probabilità identiche è noto come **principio di indifferenza**. Tale principio è spesso attribuito a Laplace (1816), che tuttavia non utilizzò mai tale termine in modo esplicito, a differenza di Keynes (1921). George Boole e John Venn vi fecero riferimento entrambi come **principio di ragione insufficiente**.

Il dibattito tra oggettivisti e soggettivisti si fece più aspro nel XX secolo. Kolmogorov (1963), R. A. Fisher (1922) e Richard von Mises (1928) difesero l'interpretazione della frequenza relativa. L'interpretazione della “propensione” di Karl Popper (1959, pubblicato inizialmente in tedesco nel 1934) fa risalire le frequenze relative a un simmetria fisica sottostante. Frank Ramsey (1931), Bruno de Finetti (1937), R. T. Cox (1946), Leonard Savage (1954), Richard Jeffrey (1983) ed E. T. Jaynes (2003) interpretarono le probabilità come i gradi di credenza di individui specifici. Le loro analisi dei gradi di credenza erano strettamente legate alle utilità e al comportamento, in particolar modo alla disponibilità a scommettere sugli esiti.

Rudolf Carnap fornì una diversa interpretazione soggettiva della probabilità: non come il grado di credenza che un individuo effettivamente ha, ma come il grado di credenza che un individuo idealizzato che ragiona *dovrebbe* riporre in una particolare proposizione *a* dato un particolare insieme di evidenze *e*. Carnap cercò di rendere questa nozione di **grado di conferma** matematicamente precisa, come relazione logica tra *a* ed *e*. Attualmente si ritiene che non esista una logica unica di questo tipo, ma che tale logica poggi su una distribuzione di probabilità a priori soggettiva il cui effetto diminuisce all'aumentare delle osservazioni raccolte.

Lo studio di questa relazione andò a costituire una disciplina matematica chiamata **logica induttiva**, analoga alla classica logica deduttiva (Carnap, 1948, 1950). Carnap non riuscì a estendere la logica induttiva oltre al caso proposizionale, e Putnam (1963) mostrò con argomenti a confutazione che esistevano alcuni limiti di fondo. Lavori più recenti di Bacchus, Grove, Halpern e Koller (1992) hanno esteso i metodi di Carnap a teorie del primo ordine. Il primo inqua-

drammento assiomatico rigoroso per la teoria della probabilità fu proposto da Kolmogorov (1950, prima pubblicazione in tedesco nel 1933). Rényi (1970) in seguito fornì una presentazione assiomatica che considerava primitiva la probabilità condizionale anziché la probabilità assoluta.

Oltre alle argomentazioni di de Finetti a favore della validità degli assiomi, Cox (1946) mostrò che ogni sistema di ragionamento incerto che soddisfi il suo insieme di ipotesi è equivalente alla teoria delle probabilità. Questo diede nuovo slancio ai sostenitori della probabilità, ma altri non erano convinti e obiettavano contro l’ipotesi che la credenza dovesse essere rappresentata da un singolo numero. Halpern (1999) ha descritto le ipotesi e ha mostrato alcune lacune nella formulazione originale di Cox. Horn (2003) ha mostrato come superare le difficoltà. Jaynes (2003) ha presentato un’argomentazione simile ma più facile da comprendere.

Il Rev. Thomas Bayes (1702–1761) introdusse la regola per il ragionamento su probabilità condizionali che dopo la sua morte prese il suo nome (Bayes, 1763). Bayes considerò soltanto il caso della probabilità a priori uniformi; fu Laplace a sviluppare in modo indipendente il caso generale. Il ragionamento probabilistico bayesiano è stato impiegato nell’IA fin dagli anni 1960, specialmente nella diagnosi medica. Il suo utilizzo non si limitava alla formulazione di diagnosi a partire dalle evidenze disponibili, ma anche alla selezione di ulteriori investigazioni ed esami (sfruttando la teoria del valore dell’informazione, che vedremo nel Paragrafo 16.6) quando le evidenze disponibili erano insufficienti (Gorry, 1968; Gorry *et al.*, 1973). Un sistema fu capace di superare gli esperti umani nella diagnosi delle malattie addominali acute (de Dombal *et al.*, 1974). Lucas *et al.* (2004) forniscono una panoramica.

Questi primi sistemi bayesiani avevano molte limitazioni: dato che non possedevano alcun modello teorico delle condizioni che dovevano diagnosticare, erano vulnerabili ai casi in cui i dati non fossero rappresentativi perché era disponibile un campione

tropo piccolo (de Dombal *et al.*, 1981). Un secondo problema, ancor più grave, era la mancanza di un formalismo conciso (come quello che descriveremo nel Capitolo 13) per la rappresentazione e l’utilizzo di informazione riguardante l’indipendenza condizionale. Per questo motivo erano costretti a dipendere dall’acquisizione, la memorizzazione e l’elaborazione di enormi tabelle di dati probabilistici. A causa di queste difficoltà, i metodi probabilistici per la gestione dell’incertezza caddero in disgrazia per una quindicina d’anni a partire dai primi anni 1970 fino a metà degli anni 1980. Nel prossimo capitolo descriveremo gli sviluppi ottenuti dalla fine degli anni 1980.

Il modello di Bayes ingenuo per le distribuzioni congiunte è stato studiato estensivamente dalla comunità del riconoscimento di pattern fin dagli anni 1950 (Duda e Hart, 1973). È stato anche utilizzato, spesso senza rendersene conto, per il recupero di informazioni, a partire dal lavoro di Maron (1961). I fondamenti probabilistici di questa tecnica, descritti nell’Esercizio 12.BAYS, furono sviluppati da Robertson e Sparck Jones (1976). Domingos e Pazzani (1997) spiegano il sorprendente successo del ragionamento bayesiano ingenuo anche in domini in cui le ipotesi di indipendenza sono palesemente violate.

Ci sono molti buoni testi introduttivi sulla teoria della probabilità, tra cui citiamo quelli di Bertsekas e Tsitsiklis (2008), Ross (2015), Grinstead e Snell (1997). DeGroot e Schervish (2001) offrono un’introduzione congiunta a probabilità e statistica da un punto di vista bayesiano, mentre Walpole *et al.* (2016) offrono un’introduzione per scienziati e ingegneri. Jaynes (2003) fornisce una presentazione molto convincente dell’approccio bayesiano. Billingsley (2012) e Venkatesh (2012) forniscono ulteriori trattazioni matematiche, comprendendo tutte le complicazioni relative alle variabili continue che non abbiamo trattato qui. Hacking (1975) e Hald (1990) trattano la storia del concetto di probabilità, mentre Bernstein (1996) fornisce un resoconto popolare.

CAPITOLO

13

Ragionamento probabilistico

- 13.1 Rappresentazione della conoscenza in un dominio incerto
- 13.2 La semantica delle reti bayesiane
- 13.3 Inferenza esatta nelle reti bayesiane
- 13.4 Inferenza approssimata nelle reti bayesiane
- 13.5 Reti causali
- 13.6 Riepilogo
Note storiche e bibliografiche

In cui spieghiamo come costruire modelli reticolari efficienti per ragionare in condizioni di incertezza secondo le regole della teoria della probabilità, e come distinguere tra correlazione e causalità.

Nel Capitolo 12 abbiamo presentato gli elementi di base della teoria della probabilità, evidenziando l'importanza delle relazioni di indipendenza assoluta e condizionale per semplificare le rappresentazioni probabilistiche del mondo. Questo capitolo introduce un approccio sistematico per rappresentare esplicitamente tali relazioni sotto forma di **reti bayesiane**. Ne definiremo la sintassi e la semantica, e mostreremo come possono essere usate per esprimere conoscenza incerta in modo compatto e naturale. Vedremo quindi com'è possibile svolgere in modo efficiente l'inferenza probabilistica, che nel caso pessimo è computazionalmente intrattabile, in diverse situazioni pratiche. Inoltre descriveremo alcuni algoritmi di inferenza approssimati che si possono utilizzare in molti casi in cui l'inferenza esatta non è applicabile. Nel Capitolo 15 estenderemo i concetti di base delle reti bayesiane a linguaggi formali maggiormente espressivi per la definizione di modelli di probabilità.

13.1 Rappresentazione della conoscenza in un dominio incerto

Nel capitolo precedente abbiamo visto che la distribuzione di probabilità congiunta completa può rispondere a qualsiasi domanda riguardante il dominio, ma diventa intrattabilmente grande al crescere del numero di variabili. Inoltre, specificare le probabilità per i singoli mondi possibili procedendo uno per uno è innaturale e noioso.

rete bayesiana

Abbiamo anche visto che le relazioni di indipendenza assoluta e condizionale tra le variabili possono ridurre drasticamente il numero di probabilità che devono essere specificate esplicitamente per definire la distribuzione congiunta completa. Questo paragrafo introduce una struttura dati chiamata **rete bayesiana**¹ per rappresentare le dipendenze tra le variabili. Le reti bayesiane possono rappresentare in sostanza *qualsiasi* distribuzione di probabilità congiunta completa, e in molti casi possono farlo in maniera molto concisa.

Una rete bayesiana è un grafo orientato in cui ogni *nodo* è etichettato con informazione probabilistica quantitativa. La specifica completa è la seguente.

1. Ogni nodo della rete corrisponde a una variabile casuale, che può essere discreta o continua.
2. Archi orientati (frecce) collegano coppie di nodi. Se c'è una freccia dal nodo X al nodo Y , si dice che X è *genitore* di Y . Il grafo non ha cicli diretti e quindi è un grafo aciclico orientato, o DAG (*directed acyclic graph*).
3. Ogni nodo X_i è associato a informazioni di probabilità $\theta(X_i|Genitori(X_i))$ che quantificano gli effetti dei genitori su un nodo usando un numero finito di **parametri**.

parametro

La topologia della rete, ovvero l'insieme dei nodi e quello degli archi, specifica le condizioni di indipendenza nel modo che preciseremo tra poco. Il significato *intuitivo* di una freccia, in una rete costruita correttamente, è generalmente che X ha una *influenza diretta* su Y , il che suggerisce che le cause dovrebbero essere genitori degli effetti. In generale è abbastanza facile, per un esperto, decidere quali influenze dirette esistono nel dominio; molto più facile che indicare effettivamente le probabilità. Una volta delineata la topologia di una rete bayesiana, rimane da specificare solo l'informazione di probabilità locale di ogni variabile, nella forma di una distribuzione di probabilità condizionata dati i suoi genitori. La distribuzione congiunta completa di tutte le variabili è definita dalla topologia e dall'informazione di probabilità locale.

Ricorderete il semplice mondo descritto nel Capitolo 12, che consisteva nelle sole variabili *MalDiDenti*, *Carie*, *Prende* e *CondizioniAtmosferiche*. Abbiamo affermato che *CondizioniAtmosferiche* è indipendente dalle altre variabili; inoltre *MalDiDenti* e *Prende* sono condizionalmente indipendenti, data *Carie*. Queste relazioni sono rappresentate dalla struttura della rete bayesiana mostrata nella Figura 13.1. Formalmente, l'indipendenza condizionale di *MalDiDenti* e *Prende* data *Carie* è indicata dall'assenza di un collegamento tra *MalDiDenti* e *Prende*. Intuitivamente, la rete rappresenta il fatto che *Carie* è una causa diretta di *MalDiDenti* e *Prende*, mentre non c'è alcuna relazione causale diretta tra *MalDiDenti* e *Prende*.

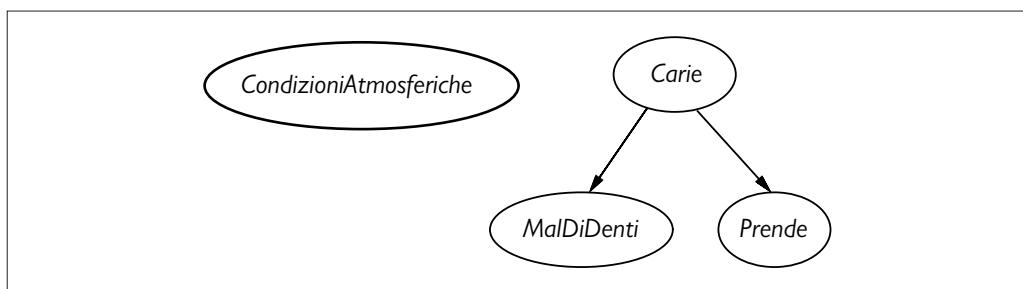


Figura 13.1 Una semplice rete bayesiana in cui *CondizioniAtmosferiche* è indipendente dalle altre tre variabili mentre *MalDiDenti* e *Prende* sono condizionalmente indipendenti data *Carie*.

¹ Le reti bayesiane, o “reti di Bayes”, venivano chiamate **reti di conoscenza** negli anni 1980 e 1990. Una **rete causale** è una rete bayesiana con vincoli aggiuntivi sul significato delle frecce (cfr. Paragrafo 13.5). Il termine **modello grafico** si riferisce a una classe più ampia che comprende al suo interno le reti bayesiane.

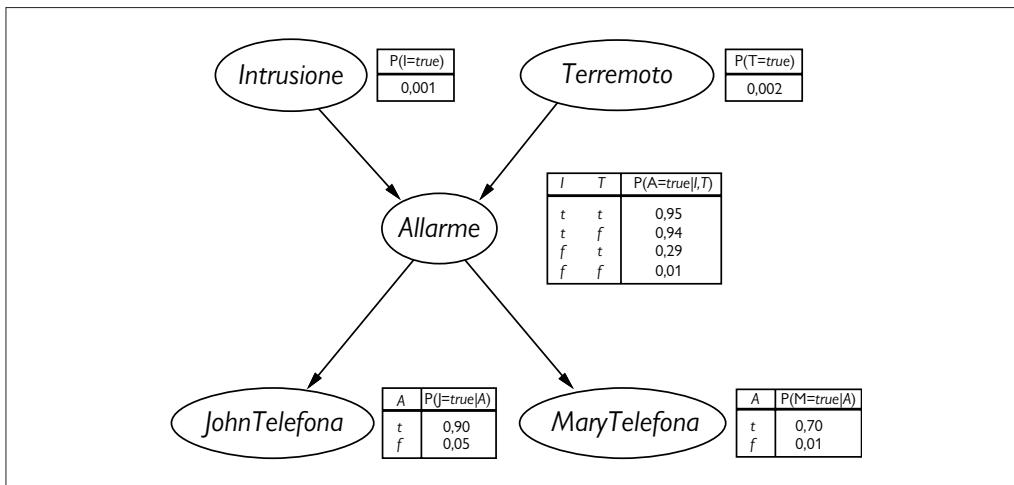


Figura 13.2
Una tipica rete bayesiana, con l'indicazione della topologia e le tabelle delle probabilità condizionate (CPT). Nelle CPT, le lettere I, T, A, J e M stanno rispettivamente per *Intrusione*, *Terremoto*, *Allarme*, *JohnTelefona* e *MaryTelefona*.

Ora considerate il seguente esempio, appena più complesso. Avete da poco installato in casa un nuovo antifurto. È abbastanza affidabile, ma occasionalmente scatta anche per piccoli terremoti (l'esempio è stato inventato da Judea Pearl, che vive a Los Angeles, in zona sismica). Avete anche due vicini, John e Mary, che hanno promesso di telefonarvi al lavoro quando sentono suonare l'allarme. John quando sente l'allarme chiama quasi sempre, ma qualche volta si confonde e scambia lo squillo del telefono in casa vostra per l'antifurto. Mary, d'altro canto, ama ascoltare musica ad alto volume e spesso non sente suonare l'allarme. Prendendo come evidenze le telefonate dei vicini, vorremmo stimare la probabilità di un'intrusione.

Una rete bayesiana per questo dominio è riportata nella Figura 13.2. La struttura della rete mostra che i furti e i terremoti influenzano direttamente la probabilità che scatti l'allarme, ma le telefonate di John e Mary dipendono esclusivamente da quest'ultimo. La rete rappresenta così le nostre ipotesi che i vicini non percepiscano direttamente i furti, non notino i terremoti di piccola entità e non si consultino prima di telefonare.

L'informazione di probabilità locale associata a ogni nodo nella Figura 13.2 assume la forma di una **tabella delle probabilità condizionate**, o CPT (*conditional probability table*). Questo tipo di tabella può essere usato solo per le variabili discrete; altre rappresentazioni, tra cui quelle adatte a variabili continue, sono descritte nel Paragrafo 13.2. Ogni riga di una CPT contiene la probabilità condizionata di ogni valore del nodo per **caso condizionante**. Un caso condizionante è semplicemente una possibile combinazione dei valori dei nodi genitori: una sorta di mondo possibile in miniatura, per così dire. Ogni riga deve avere somma 1, perché i suoi elementi rappresentano un insieme esaustivo di casi per la variabile. Per variabili booleane, una volta appurato che il valore è vero con probabilità p , la probabilità che sia falso dev'essere per forza $1 - p$, ragion per cui spesso si omette il secondo numero, come nella Figura 13.2. In generale, la tabella di una variabile booleana con k genitori booleani contiene 2^k probabilità indipendentemente specificabili. Un nodo senza genitori ha una sola riga, che riporta le probabilità a priori di ogni suo possibile valore.

Notate che la rete non ha nodi che corrispondono al fatto che Mary stia ascoltando la musica, o che lo squillo del telefono confonda John. Questi fattori sono riassunti nell'incertezza associata ai collegamenti da **Allarme** a **JohnTelefona** e **MaryTelefona**. Ciò mette in evidenza pigrizia e ignoranza nell'operatività, come si è spiegato nel Paragrafo 12.1.1 del Capitolo 12: servirebbe parecchio lavoro per determinare quali fattori sarebbero più o meno probabili in ogni particolare caso, e comunque non conosciamo un modo ragionevole per ottenere le informazioni rilevanti.

tabella delle probabilità condizionate

caso condizionante

Le probabilità riassumono, di fatto, un insieme *potenzialmente infinito* di circostanze in cui l'allarme potrebbe non scattare quando dovrebbe (umidità altissima, blackout, batterie difettose, cavi tagliati, un topo morto incastrato nella campanella, ecc.) o i vicini potrebbero non telefonare per segnalarlo (perché andati al ristorante, in vacanza, sordità temporanea, passaggio di un elicottero, ecc.). In questo modo, un piccolo agente può cavarsela in un grande mondo, almeno approssimativamente.

13.2 La semantica delle reti bayesiane

La *sintassi* di una rete bayesiana è costituita da un grafo aciclico orientato con alcune informazioni di probabilità locale assegnate a ogni nodo. La *semantica* definisce il modo in cui la sintassi corrisponde a una distribuzione congiunta sulle variabili della rete.

Supponiamo che la rete contenga n variabili, X_1, \dots, X_n . Allora un elemento generico nella distribuzione congiunta è $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$, o in forma abbreviata $P(x_1, \dots, x_n)$. La semantica della rete bayesiana definisce ogni elemento nella distribuzione congiunta come segue:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta(x_i \mid \text{genitori}(X_i)), \quad (13.1)$$

dove $\text{genitori}(X_i)$ denota i valori di $\text{Genitori}(X_i)$ che appaiono in x_1, \dots, x_n . Così, ogni elemento della distribuzione congiunta è rappresentato dal prodotto degli elementi appropriati nelle distribuzioni condizionate locali della rete bayesiana.

Per illustrare questo, possiamo calcolare la probabilità che l'allarme scatti senza che si siano verificati né un'intrusione né un terremoto, e che in conseguenza a ciò sia John che Mary telefonino. Vasta moltiplicare gli elementi rilevanti delle distribuzioni condizionate locali (abbreviamo i nomi delle variabili):

$$\begin{aligned} P(j, m, a, \neg i, \neg t) &= P(j|a)P(m|a)P(a|\neg i \wedge \neg t)P(\neg i)P(\neg t) \\ &= 0,90 \times 0,70 \times 0,001 \times 0,999 \times 0,998 = 0,000628. \end{aligned}$$

Nel Paragrafo 12.3 abbiamo spiegato come si può usare la distribuzione congiunta completa per rispondere a qualsiasi interrogazione sul dominio. Se è vero che una rete bayesiana è una rappresentazione della distribuzione congiunta, anch'essa potrà essere usata per rispondere a qualsiasi query, semplicemente sommando tutti i valori di probabilità rilevanti della distribuzione. Nel Paragrafo 13.3 spiegheremo questo in maggior dettaglio, ma descriveremo anche metodi molto più efficienti.

Finora abbiamo sorvolato su un punto importante: che significato hanno i numeri che entrano nelle distribuzioni condizionate locali $\theta(x_i \mid \text{genitori}(X_i))$? Dall'Equazione (13.1) si può dimostrare che i parametri $\theta(x_i \mid \text{genitori}(X_i))$ sono esattamente le probabilità condizionate $P(x_i \mid \text{genitori}(X_i))$ implicate dalla distribuzione congiunta. Ricordate che le probabilità condizionate possono essere calcolate dalla distribuzione congiunta come segue:

$$\begin{aligned} P(x_i \mid \text{genitori}(X_i)) &\equiv \frac{P(x_i, \text{genitori}(X_i))}{P(\text{genitori}(X_i))} \\ &= \frac{\sum_{\mathbf{y}} P(x_i, \text{genitori}(X_i), \mathbf{y})}{\sum_{x'_i, \mathbf{y}} P(x'_i, \text{genitori}(X_i), \mathbf{y})} \end{aligned}$$

dove \mathbf{y} rappresenta i valori di tutte le variabili diverse da X_i e dai suoi genitori. Dall'ultima riga si può dimostrare che $P(x_i \mid \text{genitori}(X_i)) = \theta(x_i \mid \text{genitori}(X_i))$ (Esercizio 13.CPTE). Possiamo quindi riscrivere l'Equazione (13.1) come segue:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{genitori}(X_i)). \quad (13.2)$$

Questo significa che, quando stimiamo i valori per le distribuzioni condizionate locali, questi devono essere le probabilità condizionate effettive per la variabile dati i suoi genitori. Così, per esempio, quando specifichiamo $\theta(\text{JohnTelefona} = \text{true} | \text{Allarme} = \text{true}) = 0,90$, questo significa che in circa il 90% dei casi in cui l'allarme suona, John telefona. Il fatto che ogni parametro della rete ha un significato preciso in termini di un piccolo insieme di variabili è fondamentale per la robustezza e la facilità con la quale questi modelli sono specificati.

Un metodo per costruire reti bayesiane

L'Equazione (13.2) definisce il significato di una rete bayesiana. Il prossimo passo è spiegare come *costruire* una rete bayesiana in modo che la distribuzione congiunta risultante sia una buona rappresentazione di un dato dominio. Ora vedremo che l'Equazione (13.2) implica delle relazioni di indipendenza condizionale che possono guidare l'ingegnere della conoscenza nella costruzione della topologia della rete. Prima di tutto riscriviamo gli elementi della distribuzione congiunta in termini di probabilità condizionata, usando la regola del prodotto (cfr. Equazione (12.4) nel Capitolo 12):

$$P(x_1, \dots, x_n) = P(x_n | x_{n-1}, \dots, x_1)P(x_{n-1} | x_{n-2}, \dots, x_1) \dots P(x_2 | x_1)P(x_1)$$

Poi ripetiamo il processo, riducendo ogni probabilità congiunta a una probabilità condizionata e una probabilità congiunta su un insieme più piccolo di variabili. Alla fine otteniamo un unico grande prodotto:

$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_n | x_{n-1}, \dots, x_1)P(x_{n-1} | x_{n-2}, \dots, x_1) \dots P(x_2 | x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i | x_{i-1}, \dots, x_1). \end{aligned}$$

Questa identità è chiamata **regola della catena** (*chain rule*) e vale per ogni insieme di variabili casuali. Confrontandola con l'Equazione (13.2) vediamo che la specifica della distribuzione congiunta è equivalente all'asserzione generale che, per ogni variabile X_i nella rete,

$$\mathbf{P}(X_i | X_{i-1}, \dots, X_1) = \mathbf{P}(X_i | \text{Genitori}(X_i)), \quad (13.3)$$

a patto che $\text{Genitori}(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$. Quest'ultima condizione è soddisfatta numerando i nodi in **ordine topologico**, cioè in qualsiasi ordine consistente con l'ordinamento parziale implicito nella struttura del grafo orientato. Per esempio, i nodi della Figura 13.2 potrebbero essere ordinati come $I, T, A, J, M; T, I, A, J, M$; e così via.

regola della catena

ordine topologico

Il senso dell'Equazione (13.2) è che la rete bayesiana è una rappresentazione corretta del dominio solo se ogni nodo è condizionalmente indipendente dagli altri suoi predecessori nell'ordinamento dei nodi, dati i suoi genitori. Possiamo soddisfare questa condizione procedendo come segue.

1. *Nodi*: per prima cosa determiniamo l'insieme delle variabili richieste per il modello del dominio. Poi le ordiniamo, $\{X_1, \dots, X_n\}$. Va bene qualsiasi ordinamento, ma la rete risultante sarà più compatta se le variabili vengono ordinate in modo che le cause precedano gli effetti.

2. *Collegamenti*: per i che va da 1 a n procediamo come segue:

- scegliamo un insieme minimo di genitori per X_i da X_1, \dots, X_{i-1} , tale che sia soddisfatta l'Equazione (13.3);
- per ogni genitore inseriamo un arco dal genitore a X_i ;
- scriviamo la tabella delle probabilità condizionate, $\mathbf{P}(X_i | \text{Genitori}(X_i))$.



Intuitivamente, i genitori del nodo X_i dovrebbero comprendere tutti i nodi in X_1, \dots, X_{i-1} che *influenzano direttamente* X_i . Per esempio, supponiamo di aver completato la rete nella Figura 13.2 tranne che per la scelta dei genitori di *MaryTelefona*. *MaryTelefona* è certamente influenzata dal verificarsi di una *Intrusione* o di un *Terremoto*, ma non *direttamente*. Intuitivamente, la nostra conoscenza del dominio ci dice che questi eventi influenzano il comportamento di Mary solo attraverso il loro effetto sull'allarme antifurto. Inoltre, dato lo stato dell'allarme, il fatto che John telefoni o meno non ha alcuna influenza su Mary. Formalmente, diciamo che è verificato il seguente enunciato di indipendenza condizionale:

$$\mathbf{P}(\text{MaryTelefona} | \text{JohnTelefona}, \text{Allarme}, \text{Terremoto}, \text{Intrusione}) = \mathbf{P}(\text{MaryTelefona} | \text{Allarme}).$$

Quindi, *Allarme* sarà l'unico nodo genitore per *MaryTelefona*.

Poiché ogni nodo è connesso soltanto ai precedenti, questo metodo di costruzione garantisce che la rete sia aciclica. Un'altra importante proprietà delle reti bayesiane è che non contengono valori di probabilità ridondanti. Se non vi è ridondanza, non vi è spazio per inconsistenza: è *impossibile per l'ingegnere della conoscenza o l'esperto del dominio creare una rete bayesiana che violi gli assiomi della teoria delle probabilità*.



**localmente strutturato
sparso**

Compattezza e ordinamento dei nodi

Oltre a essere una rappresentazione completa e non ridondante del dominio, una rete bayesiana può essere molto più *compatta* della distribuzione congiunta completa. Questa proprietà è ciò che rende possibile gestire domini con molte variabili. La compattezza delle reti bayesiane è un esempio di una proprietà generale dei sistemi **localmente strutturati** (chiamati anche sistemi **sparsi**): in un sistema localmente strutturato, ogni sottocomponente interagisce direttamente solo con pochi altri componenti, indipendentemente dal loro numero totale. La strutturazione locale solitamente fa sì che la crescita della complessità sia lineare anziché esponenziale.

Nel caso delle reti bayesiane, è ragionevole supporre che nella maggior parte dei domini ogni variabile casuale sia influenzata direttamente da al più k altre, per un valore costante di k . Se consideriamo per semplicità n variabili booleane, la quantità di informazione necessaria per specificare ogni tabella delle probabilità condizionate sarà costituita al più da $2^k \cdot n$ numeri, e la rete completa potrà essere specificata con $n2^k$ numeri. La distribuzione congiunta, d'altra parte, contiene 2^n numeri. Per fare un esempio concreto, supponiamo di avere $n = 30$ nodi, ognuno con cinque genitori ($k = 5$). In questo caso la rete bayesiana richiede 960 numeri, la distribuzione congiunta completa oltre un miliardo.

Specificare le tabelle delle probabilità condizionate per una rete completamente connessa, in cui ogni variabile ha tutti i suoi predecessori come genitori, richiede la stessa quantità di informazioni di specificare la distribuzione congiunta in forma tabellare. Per questo motivo, spesso lasciamo fuori dei collegamenti anche se esiste una lieve dipendenza, dato che il piccolo guadagno di accuratezza non vale la maggiore complessità della rete. Per esempio, si potrebbe criticare la nostra rete antifurto osservando che, in caso di forte terremoto, John e Mary potrebbero non chiamare anche dopo aver sentito l'allarme, presumendo che sia scattato a causa dei tremori. La decisione di aggiungere un collegamento da *Terremoto* a *JohnTelefona* e *MaryTelefona* (aumentando così la dimensione delle tabelle) dipende dall'importanza relativa di ottenere probabilità più accurate rispetto al costo di specificare l'informazione aggiuntiva.

Anche in un dominio localmente strutturato, otterremo una rete bayesiana compatta soltanto se scegliamo bene l'ordinamento dei nodi. Che cosa accade se scegliamo un ordinamento sbagliato? Consideriamo ancora l'esempio dell'antifurto. Supponiamo di aver deciso di aggiungere i nodi nell'ordine *MaryTelefona*, *JohnTelefona*, *Allarme*, *Intrusione*, *Terremoto*. Quella che otteniamo è la rete, leggermente più complicata, mostrata nella Figura 13.3(a). Il processo si svolge come segue.

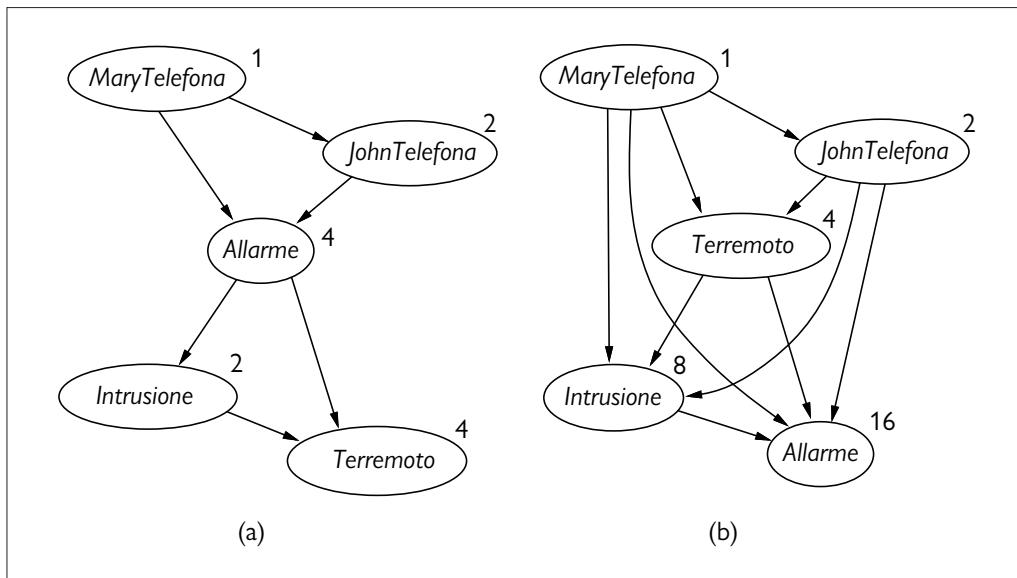


Figura 13.3 La struttura di una rete e il numero dei parametri dipendono dall'ordine di introduzione. (a) La struttura ottenuta con l'ordinamento M, J, A, B, E . (b) La struttura ottenuta con l'ordinamento M, J, E, B, A . Ogni nodo è etichettato con il numero di parametri richiesti: sono 13 in tutto per (a) e 31 per (b). Nella struttura della Figura 13.2 erano richiesti soltanto 10 parametri.

- Aggiunto *MaryTelefona*: nessun genitore.
 - Aggiunto *JohnTelefona*: se Mary telefona, probabilmente è perché l'allarme è scattato, il che rende più probabile una chiamata da parte di John. Quindi *JohnTelefona* ha *MaryTelefona* come padre.
 - Aggiunto *Allarme*: chiaramente, se entrambi i vicini telefonano è più probabile che l'allarme sia scattato rispetto al caso in cui telefona uno solo o nessuno, per cui questo nodo avrà *MaryTelefona* e *JohnTelefona* come genitori.
 - Aggiunto *Intrusione*: se conosciamo lo stato dell'allarme, una chiamata da parte di John o Mary ci potrebbe fornire informazioni sullo stato del nostro telefono o della musica di Mary, ma non riguardo a un'eventuale intrusione:

$$\mathbf{P}(Intrusione|Allarme, JohnTelefona, MaryTelefona) = \mathbf{P}(Intrusione|Allarme).$$

Quindi l'unico genitore necessario è *Allarme*.

- Aggiunto *Terremoto*: se l'allarme è scattato, è più probabile che si sia verificato un terremoto (l'antifurto, a suo modo, è una specie di rilevatore di movimenti tellurici). Ma se sappiamo che si è verificata un'intrusione, allora quella spiega l'allarme, e la probabilità di un terremoto sarebbe solo lievemente superiore a quella normale. Quindi, sia *Allarme* che *Intrusione* sono genitori.

La rete risultante ha due collegamenti in più di quella originale della Figura 13.2 e richiede 13 probabilità condizionate anziché 10. Quel che è peggio, alcuni collegamenti rappresentano relazioni molto tenui che richiedono giudizi probabilistici difficili e innaturali: pensate alla valutazione della probabilità di un *Terremoto*, dati *Intrusione* e *Allarme*. Questo fenomeno è piuttosto generale ed è collegato alla distinzione tra modelli **causal**i e **diagnostici** introdotta nel Paragrafo 12.5 (cfr. anche l'Esercizio 13.WUMD). Attenendoci al modello causale, dovremo specificare meno numeri, e spesso questi saranno più facili da determinare. Nel dominio della medicina, per esempio, è stato dimostrato da Tversky e Kahneman (1982) che i medici esperti preferiscono fornire giudizi probabilistici sulle regole causali piuttosto che su



quelle diagnostiche. Nel Paragrafo 13.5 esamineremo in maggior dettaglio il concetto di modelli causali.

La Figura 13.3(b) mostra un ordinamento dei nodi molto discutibile: *MaryTelefona, JohnTelefona, Terremoto, Intrusione, Allarme*. Questa rete, per essere specificata, richiede 31 probabilità distinte: esattamente lo stesso numero della distribuzione congiunta completa. È importante comprendere, comunque, che tutte e tre le reti rappresentano *esattamente la stessa distribuzione congiunta*. Le due versioni della Figura 13.3 non riescono a rappresentare tutte le relazioni di indipendenza condizionale e quindi sono costrette a specificare molti numeri superflui.

13.2.1 Relazioni di indipendenza condizionale nelle reti bayesiane

Dalla semantica delle reti bayesiane definita nell'Equazione (13.2) possiamo dedurre diverse proprietà di indipendenza condizionale. Abbiamo già visto la proprietà che una variabile è condizionalmente indipendente dai suoi altri predecessori dati i suoi genitori; è anche possibile dimostrare la proprietà più generale sui “non discendenti”:

discendente

Ogni variabile è condizionalmente indipendente dai suoi non discendenti, dati i suoi genitori.

Per esempio, nella Figura 13.2 la variabile *JohnTelefona* è indipendente da *Intrusione, Terremoto* e *MaryTelefona* dato il valore di *Allarme*. La definizione è illustrata nella Figura 13.4(a).

La proprietà dei non discendenti combinata con l'interpretazione dei parametri di rete $\theta(X_i | \text{Genitori}(X_i))$ come probabilità condizionate $\mathbf{P}(X_i | \text{Genitori}(X_i))$ è sufficiente per ricostruire la distribuzione congiunta completa fornita nell'Equazione (13.2). In altre parole, la semantica delle reti bayesiane può essere vista in un modo diverso: anziché definire la distribuzione congiunta completa come prodotto di distribuzioni condizionate, la rete definisce un insieme di proprietà di indipendenza condizionale e la distribuzione congiunta completa può essere ricavata da queste.

La proprietà dei non discendenti implica anche un'altra importante proprietà di indipendenza:

coperta di Markov

Una variabile è condizionalmente indipendente da tutti gli altri nodi della rete dati i suoi genitori, i suoi figli e i genitori dei figli, cioè data la sua coperta di Markov.

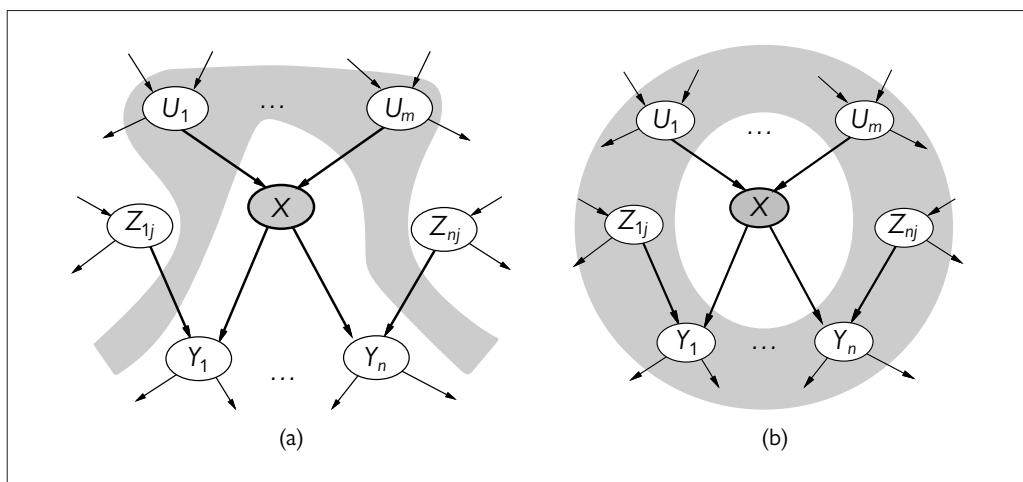


Figura 13.4 (a) Un nodo X è condizionalmente indipendente dai suoi non-discendenti (nella figura, i nodi Z_{ij}) dati i suoi genitori (gli U_i evidenziati dall'area in grigio). (b) Un nodo X è condizionalmente indipendente da tutti gli altri nodi nella rete data la sua coperta di Markov (l'area evidenziata in grigio).

Nell’Esercizio 13.MARB viene richiesto di dimostrarlo. Per esempio, la variabile *Intrusione* è indipendente da *JohnTelefona* e *MaryTelefona*, dati *Allarme* e *Terremoto*. Questa proprietà è illustrata nella Figura 13.4(b). La proprietà della coperta di Markov consente di sviluppare algoritmi di inferenza che usano processi di campionamento stocastico completamente locali e distribuiti, come viene spiegato nel Paragrafo 13.4.2.

La domanda più generale sull’indipendenza condizionale che si potrebbe porre in una rete bayesiana è se un insieme di nodi **X** sia condizionalmente indipendente da un altro insieme **Y**, dato un terzo insieme **Z**. Lo si può determinare in modo efficiente esaminando la rete per verificare se **Z** d-separa **X** e **Y**. La procedura è descritta di seguito.

1. Considerare soltanto il **sottografo ancestrale** costituito da **X**, **Y**, **Z** e dai loro ascendenti.
2. Aggiungere archi tra qualsiasi coppia di nodi non collegati che condivide un figlio comune; così otteniamo il cosiddetto **grafo morale**.
3. Sostituire tutti gli archi orientati con archi non orientati.
4. Se **Z** blocca tutti i cammini tra **X** e **Y** nel grafo risultante, allora **Z** d-separa **X** e **Y**. In tal caso, **X** è condizionalmente indipendente da **Y**, dato **Z**. Altrimenti, la rete bayesiana originale non richiede l’indipendenza condizionale.

In breve, la d-separazione significa separazione nel sottografo non orientato, moralizzato, ancestrale. Applicando la definizione alla rete della Figura 13.2, possiamo dedurre che *Intrusione* e *Terremoto* sono indipendenti dato l’insieme vuoto (cioè sono assolutamente indipendenti); che *non* sono necessariamente condizionalmente indipendenti dato *Allarme*; e che *JohnTelefona* e *MaryTelefona* sono condizionalmente indipendenti dato *Allarme*. Noteate anche che la proprietà della coperta di Markov segue direttamente dalla proprietà di d-separazione, poiché la coperta di Markov di una variabile d-separa quest’ultima da tutte le altre variabili.

13.2.2 Rappresentazione efficiente delle distribuzioni condizionate

Anche nel caso in cui il numero massimo di genitori k sia piuttosto piccolo, riempire la tabella delle probabilità condizionate di un nodo richiede fino a $O(2^k)$ numeri e potenzialmente una grande esperienza di tutti i possibili casi condizionanti. Questo tuttavia è un caso pessimo, in cui la relazione tra il nodo figlio e i suoi genitori è completamente arbitraria. Di solito tali relazioni si possono descrivere con una **distribuzione canonica** che segue uno schema standard. In questi casi, la tabella completa può essere specificata indicando soltanto il nome dello schema e forse qualche parametro.

L’esempio più semplice è rappresentato dai **nodi deterministici**. Il valore di un nodo deterministico è specificato esattamente da quello dei suoi genitori, senza alcuna incertezza. La relazione potrebbe essere logica: per esempio, la relazione tra i genitori *Canadese*, *Statunitense* e *Messicano* e il nodo figlio *NordAmerican* sta nel fatto che il figlio è la disunione dei genitori. La relazione potrebbe anche essere numerica: per esempio, il *MigliorPrezzo* per un’automobile è il minimo dei prezzi praticati dai rivenditori della zona; l’*AcquaImmagazzinata* in un bacino idrico a fine anno è la somma della quantità di partenza più i flussi in entrata (fiumi, ruscellamento, precipitazioni) meno i flussi in uscita (rilasci, evaporazione, infiltrazioni).

Molti sistemi basati su reti bayesiane consentono all’utente di specificare funzioni deterministiche utilizzando un linguaggio di programmazione generico; in questo modo è possibile includere elementi complessi come i modelli climatici globali o i simulatori di rete energetica all’interno di un modello probabilistico.

Un altro importante schema che si incontra spesso nella pratica è l’**indipendenza specifica del contesto**, o CSI (*context-specific independence*). Una distribuzione condizionata presenta

d-separazione

**sottografo
ancestrale**

grafo morale

**distribuzione
canonica**

nodo deterministico

**indipendenza
specific
del contesto**

indipendenza specifica del contesto se una variabile è condizionalmente indipendente da alcuni dei suoi genitori dati *certi valori* di altre variabili. Per esempio, supponiamo che il *Danno* per la vostra auto verificatosi durante un dato periodo di tempo dipenda dalla *Robustezza* dell'auto e dal fatto che si verifichi o meno un *Incidente* in quel periodo. Naturalmente, se *Incidente* è falso, allora il *Danno*, se c'è, non dipende dalla *Robustezza* dell'auto (potrebbero esserci danni da vandalismo alla vernice o ai finestrini, ma ipotizzeremo che tutte le auto abbiano pari probabilità di subire tali danni). Diciamo che vi è indipendenza specifica del contesto tra *Danno* e *Robustezza* dato *Incidente = false*. I sistemi basati su reti bayesiane spesso implementano l'indipendenza specifica del contesto per specificare distribuzioni condizionate; per esempio, potremmo scrivere:

$$\mathbf{P}(Danno \mid Robustezza, Incidente) = \\ \mathbf{if} (Incidente = \text{false}) \mathbf{then} d_1 \mathbf{else} d_2(Robustezza)$$

dove d_1 e d_2 rappresentano distribuzioni arbitrarie. Come per il determinismo, la presenza di indipendenza specifica del contesto in una rete potrebbe facilitare un'inferenza efficiente. Tutti gli algoritmi di inferenza esatta citati nel Paragrafo 13.3 possono essere modificati per sfruttare l'indipendenza specifica del contesto allo scopo di velocizzare l'elaborazione.

OR rumoroso

Spesso le relazioni incerte possono essere caratterizzate mediante le cosiddette relazioni logiche **rumorose**. L'esempio classico è la relazione di **OR rumoroso**, una generalizzazione dell'OR logico. Nel calcolo proposizionale, potremmo dire che *Febbre* è vera se e solo se sono vere *Raffreddore*, *Influenza* o *Malaria*. Il modello dell'OR rumoroso permette di esprimere incertezza sulla capacità di ogni genitore di rendere vero il figlio: la relazione causale tra i due potrebbe essere *inibita*, e così un paziente potrebbe avere un raffreddore senza manifestazioni febbri.

nodo pozzo

Il modello si basa su due assunti: prima di tutto, che siano elencate tutte le possibili cause (ove ne mancassero alcune, sarebbe sempre possibile aggiungere un **nodo pozzo** per coprire le "altre cause" rimanenti); in secondo luogo, che l'inibizione di ogni genitore sia indipendente da quella degli altri: per esempio, ciò che impedisce a *Malaria* di causare una febbre, qualsiasi cosa possa essere, è indipendente da ciò che inibisce *Influenza* nello stesso modo. Date queste ipotesi, *Febbre* è falsa se e solo se tutti i suoi genitori veri sono inibiti, e la probabilità che questo accada è il prodotto delle probabilità di inibizione di ogni genitore. Supponiamo che le singole probabilità di inibizione siano le seguenti:

$$q_{raffreddore} = P(\neg\text{febbre} \mid \text{raffreddore}, \neg\text{influenza}, \neg\text{malaria}) = 0,6, \\ q_{raffreddore} = P(\neg\text{febbre} \mid \neg\text{raffreddore}, \text{influenza}, \neg\text{malaria}) = 0,2, \\ q_{raffreddore} = P(\neg\text{febbre} \mid \neg\text{raffreddore}, \neg\text{influenza}, \text{malaria}) = 0,1.$$

Allora, con queste informazioni e con gli assunti dell'OR rumoroso, è possibile ricostruire l'intera CPT. La regola generale è che

$$P(x_i \mid \text{genitori}(X_i)) = 1 - \prod_{\{j : X_j = \text{true}\}} q_j,$$

dove il prodotto è effettuato su tutti i genitori impostati a *true* per la riga corrispondente della CPT. La Figura 13.5 mostra i calcoli.

In generale, le relazioni logiche rumorose in cui una variabile dipende da k genitori possono essere descritte usando $O(k)$ parametri anziché gli $O(2^k)$ richiesti dalla tabella delle probabilità condizionate completa. Questo rende più facili la stima e l'apprendimento dei valori: per esempio, la rete CPCS (Pradhan *et al.*, 1994) sfrutta distribuzioni di OR rumoroso e MAX rumoroso per modellare relazioni tra malattie e sintomi nella medicina interna. Con 448 nodi e 906 collegamenti, i parametri richiesti sono solo 8254, invece dei 133.931.430 necessari per una rete con CPT complete.

Raffreddore	Influenza	Malaria	$P(\text{febbre} \cdot)$	$P(\neg\text{febbre} \cdot)$
f	f	f	0,0	1,0
f	f	t	0,9	0,1
f	t	f	0,8	0,2
f	t	t	0,98	$0,02 = 0,2 \times 0,1$
t	f	f	0,4	0,6
t	f	t	0,94	$0,06 = 0,6 \times 0,1$
t	t	f	0,88	$0,12 = 0,6 \times 0,2$
t	t	t	0,988	$0,012 = 0,6 \times 0,2 \times 0,1$

Figura 13.5 Tabella delle probabilità condizionate complete per $\mathbf{P}(\text{Febbre} | \text{Raffreddore}, \text{Influenza}, \text{Malaria})$, ipotizzando un modello di OR rumoroso con i tre valori di q mostrati in grassetto.

13.2.3 Reti bayesiane con variabili continue

Molti problemi reali coinvolgono quantità continue come altezza, massa, temperatura e denaro. Per definizione, le variabili continue hanno un numero infinito di valori possibili, dimodoché è impossibile specificare esplicitamente le probabilità condizionate di ognuno di essi. Un modo di gestire le variabili continue è di ricorrere alla **discretizzazione**, che consiste nel dividere i possibili valori in un insieme prefissato di intervalli. Per esempio, le temperature potrebbero essere divise in tre categorie: ($<0^{\circ}\text{C}$), ($0^{\circ}\text{C}-100^{\circ}\text{C}$), ($>100^{\circ}\text{C}$). Nella scelta del numero di categorie occorre puntare a un equilibrio tra perdita di accuratezza e aumento delle dimensioni delle CPT che potrebbe causare tempi di esecuzione eccessivi.

Un altro approccio consiste nel definire una variabile continua usando una delle famiglie standard di funzioni di densità di probabilità (cfr. Appendice A). Per esempio, una distribuzione gaussiana (o normale) $\mathcal{N}(x; \mu, \sigma^2)$ è specificata da soli due parametri, la media μ e la varianza σ^2 . Un'altra soluzione, talvolta detta rappresentazione **nonparametrica**, consiste nel definire implicitamente la distribuzione condizionata con una serie di istanze, ognuna delle quali contiene valori specifici delle variabili genitore e figlio. Esamineremo questo approccio in modo più approfondito nel Capitolo 19 del Volume 2.

Una rete con variabili sia discrete che continue è chiamata **rete bayesiana ibrida**. Per una rete ibrida si devono specificare due nuovi tipi di distribuzioni: quella condizionata di una variabile continua dati genitori discreti o continui e quella condizionata di una variabile discreta dati genitori continui. Considerate il semplice esempio della Figura 13.6, in cui un cliente acquista della frutta a seconda del suo costo, che a sua volta dipende dalle dimensioni del raccolto e dal fatto che siano disponibili o no i sussidi governativi. La variabile *Costo* è continua e ha genitori sia continui che discreti; la variabile *Acquista* è discreta e ha un genitore continuo.

discretizzazione

nonparametrica

rete bayesiana
ibrida

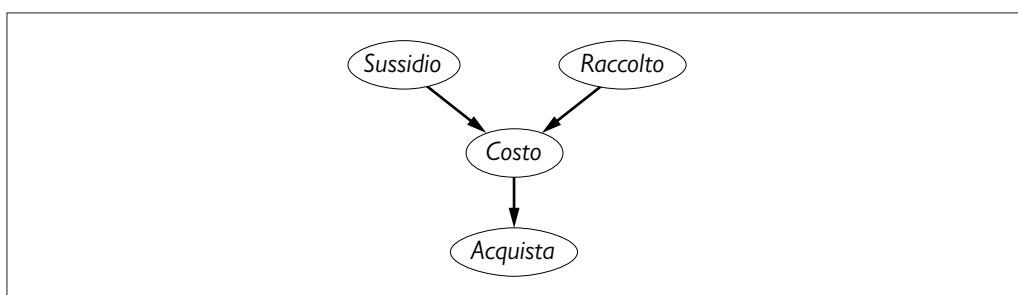


Figura 13.6
Una semplice rete con variabili discrete (Sussidio e Acquista) e continue (Raccolto e Costo).

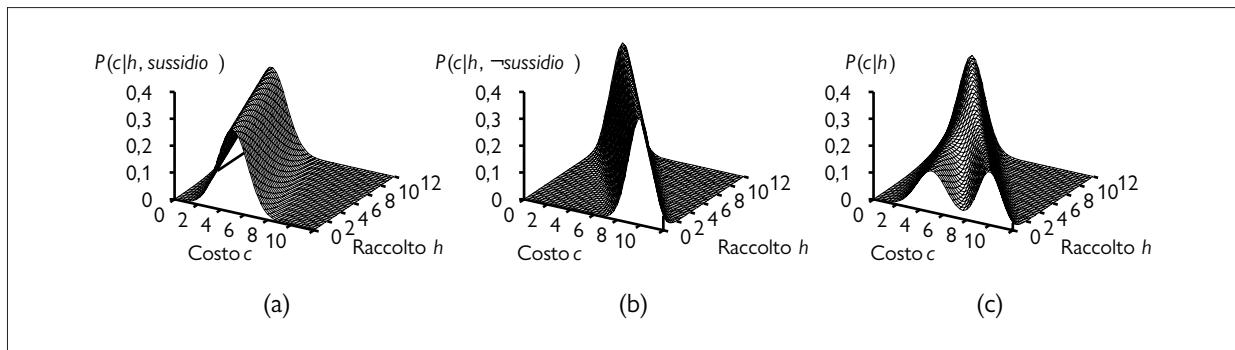


Figura 13.7 I grafici in (a) e (b) mostrano la distribuzione di probabilità di *Costo* in funzione della dimensione del *Raccolto*, rispettivamente con *Sussidio* vero e falso. Il grafico (c) mostra la distribuzione $P(\text{Costo}|\text{Raccolto})$, ottenuta sommando i due casi precedenti.

gaussiana lineare

Per la variabile *Costo* occorre specificare $\mathbf{P}(\text{Costo}|\text{Raccolto}, \text{Sussidio})$. Il genitore discreto è gestito mediante enumerazione esplicita, indicando cioè sia $\mathbf{P}(\text{Costo}|\text{Raccolto}, \text{sussidio})$ che $\mathbf{P}(\text{Costo}|\text{Raccolto}, \neg\text{sussidio})$. Per gestire *Raccolto* dobbiamo specificare in che modo la distribuzione sul costo c dipende dal valore continuo h di *Raccolto*: in altre parole, i parametri della distribuzione del costo in funzione di h . La scelta più comune è la distribuzione condizionata **gaussiana lineare**, in cui il nodo figlio ha una distribuzione gaussiana con una media μ che varia linearmente con il valore del genitore e una deviazione standard σ prefissata. Sono necessarie due distribuzioni, una per *sussidio* e una per $\neg\text{sussidio}$, con parametri differenti:

$$\begin{aligned} P(c | h, \text{sussidio}) &= \mathcal{N}(c; a_t h + b_t, \sigma_t^2) = \frac{1}{\sigma_t \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{c - (a_t h + b_t)}{\sigma_t} \right)^2} \\ P(c | h, \neg\text{sussidio}) &= \mathcal{N}(c; a_f h + b_f, \sigma_f^2) = \frac{1}{\sigma_f \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{c - (a_f h + b_f)}{\sigma_f} \right)^2}. \end{aligned}$$

In questo esempio, quindi, la distribuzione condizionata di *Costo* è specificata indicando la distribuzione gaussiana lineare e fornendo i parametri $a_t, b_t, \sigma_t, a_f, b_f$ e σ_f . Le Figure 13.7(a) e (b) illustrano queste due relazioni. Notate che in ogni caso la pendenza è di c rispetto a h negativa, perché il costo diminuisce al crescere della quantità raccolta. Naturalmente, l'ipotesi di linearità farà sì che a un certo punto il costo diventi negativo: il modello lineare è ragionevole solo se le dimensioni del raccolto variano all'interno di un intervallo ragionevolmente piccolo. La Figura 13.7(c) mostra la distribuzione $P(c|h)$, facendo la media dei due possibili valori di *Sussidio* e presumendo che ognuno di essi abbia una probabilità a priori di verificarsi di 0,5. Questo dimostra che, anche nel caso di modelli semplici, possono essere rappresentate distribuzioni piuttosto interessanti.

La distribuzione condizionata gaussiana lineare ha delle caratteristiche speciali. Una rete che contiene solo variabili continue con distribuzioni gaussiane lineari ha come distribuzione congiunta una distribuzione gaussiana multivariata (cfr. Appendice A) su tutte le variabili (cfr. Esercizio 13.5). Inoltre, anche la distribuzione a posteriori data una qualsiasi evidenza ha questa proprietà.² Quando si aggiungono variabili discrete come genitori (non

² Ne consegue che nelle reti gaussiane lineari l'inferenza richiede nel caso pessimo solo un tempo $O(n^3)$, indipendentemente dalla topologia della rete. Nel Paragrafo 13.3 vedremo che, per le reti con variabili discrete, l'inferenza è NP-difficile.

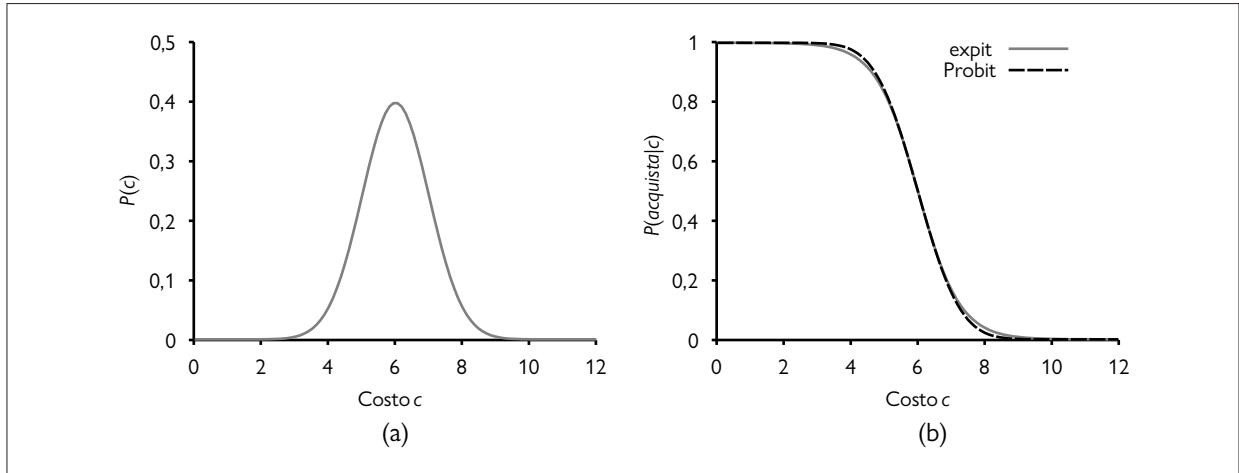


Figura 13.8 (a) Una distribuzione normale (gaussiana) per la soglia di costo, centrata in $\mu = 6,0$ con deviazione standard $\sigma = 1,0$.
(b) Modelli expit e probit per la probabilità di acquista dato costo, per i parametri $\mu = 6,0$ e $\sigma = 1,0$.

figli) di una variabile continua, la rete definisce una distribuzione **gaussiana condizionata**, o CG: dato un qualsiasi assegnamento alle variabili discrete, la distribuzione su quelle continue è una gaussiana multivariata.

Passiamo alle distribuzioni per le variabili discrete con genitori continui. Considerate per esempio il nodo *Acquista* della Figura 13.6. Sembra ragionevole presumere che il cliente acquiserà se il costo è basso e non lo farà se il costo è alto, e che la relativa probabilità varierà con continuità in un intervallo intermedio. In altre parole, la distribuzione condizionata è come una funzione soglia “morbida”. Un modo di realizzare funzioni soglia morbide è usare l'*integrale* della distribuzione normale standardizzata:

$$\Phi(x) = \int_{-\infty}^x \mathcal{N}(s; 0, 1) ds.$$

$\Phi(x)$ è una funzione crescente di x , mentre la probabilità di acquisto diminuisce con il costo, perciò ribaltiamo la funzione:

$$P(\text{acquista} | \text{Costo} = c) = 1 - \Phi((c - \mu)/\sigma),$$

questo significa che la soglia di costo si posiziona intorno alla media μ , l'ampiezza della regione di soglia è proporzionale a σ , e la probabilità di comprare diminuisce all'aumentare del costo. Questo modello **probit** (abbreviazione di *probability unit*) è illustrato nella Figura 13.8(a). La sua forma può essere giustificata supponendo che il processo decisionale sottostante abbia una soglia ben definita, la cui posizione precisa è soggetta a un rumore gaussiano casuale. Un'alternativa al modello probit è il modello **expit** o **logit inverso**, che utilizza la **funzione logistica** $1/(1+e^{-x})$ per produrre una soglia morbida: fa corrispondere ogni x a un valore tra 0 e 1. Ancora, per il nostro esempio ribaltiamo la funzione per ottenerne una decrescente, inoltre scaliamo l'esponente di $4/\sqrt{2\pi}$ per fare corrispondere la pendenza del modello probit con la media:

$$P(\text{acquista} | \text{Costo} = c) = 1 - \frac{1}{1 + \exp\left(-\frac{4}{\sqrt{2\pi}} \cdot \frac{c - \mu}{\sigma}\right)}.$$

Quest'ultima è illustrata nella Figura 13.8(b). Le due distribuzioni appaiono simili, ma la logit ha delle “code” molto più lunghe. Di solito la probit si adatta meglio alle situazioni reali, mentre la funzione logistica è talvolta più facile da trattare matematicamente. È usata am-

gaussiana condizionata

probit

expit

logit inverso

funzione logistica

piamente nell'apprendimento automatico. Entrambi i modelli possono essere generalizzati per gestire genitori continui multipli, prendendo una combinazione lineare dei loro valori. Questo metodo funziona anche per genitori discreti se i loro valori sono interi; per esempio, con k genitori booleani, ognuno visto come se avesse valore 0 o 1, l'input per la distribuzione exhit o probit sarebbe una combinazione lineare pesata con k parametri, e si otterebbe un modello abbastanza simile al modello di OR rumoroso trattato in precedenza.

13.2.4 Caso di studio: assicurazione auto

variabile nascosta

Una società di assicurazioni auto riceve da una persona la richiesta di assicurare uno specifico veicolo e deve decidere il premio annuo da applicare, in base a una previsione dei rimborsi che dovrà sostenere per quel richiedente. Occorre costruire una rete bayesiana che catturi la struttura causale del dominio e fornisca una distribuzione accurata e ben calibrata sulle variabili di output date le evidenze rese disponibili dal modulo di richiesta.³ La rete bayesiana comprenderà **variabili nascoste** che non sono né di input né di output, ma sono essenziali per strutturare la rete in modo che sia ragionevolmente sparsa con un numero di parametri gestibile. Le variabili nascoste sono raffigurate in grigio scuro nella Figura 13.9.

Le richieste di indennizzo da pagare – in grigio chiaro nella Figura 13.9 – sono di tre tipi: il *CostoMedico* per qualsiasi danno biologico subito dal richiedente; il *CostoResponsabilità* per le cause intentate da altre parti contro il richiedente e la società assicuratrice; il *CostoProprietà* per danni ad auto di ciascuna delle parti e in caso di furto. Il modulo di richiesta chiede di inserire le seguenti informazioni (nodi in bianco nella Figura 13.9):

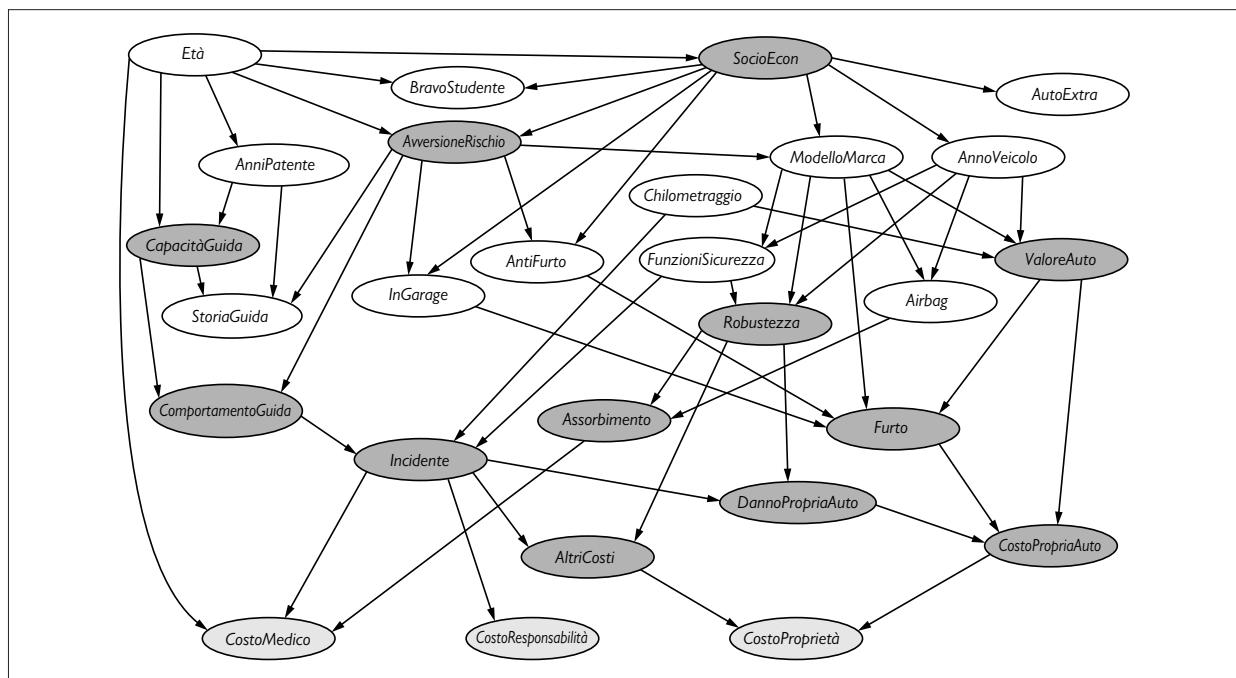


Figura 13.9 Una rete bayesiana per valutare polizze di assicurazione auto.

³ La rete illustrata nella Figura 13.9 non è realmente in uso, ma la sua struttura è stata controllata con esperti di assicurazioni. Nella pratica, le informazioni richieste sui moduli delle società assicuratrici variano da una società all'altra e in base alla giurisdizione; per esempio, alcuni chiedono il *Genere*. Il modello potrebbe certamente essere reso più dettagliato e sofisticato.

- Riguardo il richiedente: *Età; AnniPatente* – da quanto tempo ha la patente; *StoriaGuida* – un riepilogo, magari basato su “punti” di incidenti e violazioni del codice della strada commesse recentemente; inoltre (per gli studenti) un indicatore *BravoStudente* con una media a punti di 3,0 (B) su una scala di 4 punti.
- Riguardo il veicolo: *ModelloMarca* e *AnnoVeicolo*; se ha o meno un *Airbag*; e un riepilogo di *FunzioniSicurezza* come il sistema anti slittamento in frenata e quello per l'avviso in caso di pericolo di collisione.
- Riguardo la situazione di guida: il *Chilometraggio* annule e se il veicolo è *InGarage*.

Ora dobbiamo pensare a come disporre tutte queste informazioni in una struttura. Le variabili nascoste fondamentali indicano se un *Furto* o un *Incidente* si verificherà nel prossimo periodo di tempo. Naturalmente non è possibile chiedere una previsione a chi presenta la richiesta di polizza: è necessario inferire queste variabili dai dati disponibili e dalla precedente esperienza dell'assicuratore.

Quali sono i fattori causali che portano a *Furto*? Il *ModelloMarca* è certamente importante, infatti alcuni modelli sono rubati molto più spesso di altri perché c’è un efficiente mercato di rivendita per veicoli e parti di ricambio; conta anche il *ValoreAuto*, perché i veicoli vecchi, in cattive condizioni o con chilometraggio elevato hanno un valore di rivendita più basso. Inoltre, un veicolo che è *InGarage* e ha un dispositivo *AntiFurto* è più difficile da rubare. La variabile nascosta *ValoreAuto* dipende a sua volta da *ModelloMarca*, *AnnoVeicolo* e *Chilometraggio*. *ValoreAuto* determina anche la somma persa quando si verifica un *Furto*, perciò è uno dei contributori di *CostoPropriaAuto* (gli altri sono gli incidenti, che vedremo tra poco).

In modelli di questo tipo è prassi comune introdurre un’altra variabile nascosta, *SocioEcon*, la categoria socioeconomica del richiedente. Si pensa che questo influenzi un’ampia varietà di comportamenti e caratteristiche. Nel nostro modello non vi è evidenza *diretta* in forma di variabili di reddito e occupazione osservate;⁴ tuttavia *SocioEcon* influenza *ModelloMarca* e *AnnoVeicolo*; inoltre ha effetto su *AutoExtra* e *BravoStudente* e dipende in qualche modo da *Età*.

Per qualsiasi società assicurativa, la variabile nascosta forse più importante è *AvversioneRischio*: le persone avverse al rischio, infatti, sono ottimi clienti per le assicurazioni! *Età* e *SocioEcon* influiscono su *AvversioneRischio*, e tra i suoi “sintomi” vi è la scelta di *InGarage* e di dispositivi *AntiFurto* e *FunzioniSicurezza*.

L’aspetto fondamentale per la previsione di futuri incidenti è il futuro *ComportamentoGuida* del richiedente, che è influenzato sia da *AvversioneRischio* che da *CapacitàGuida*; quest’ultima a sua volta dipende da *Età* e *AnniPatente*. Il comportamento passato del richiedente si riflette in *StoriaGuida*, che a sua volta dipende da *AvversioneRischio* e *CapacitàGuida* oltre che da *AnniPatente* (perché chi ha cominciato a guidare da poco potrebbe non aver avuto il tempo di accumulare una lunga lista di incidenti e violazioni del codice della strada). In questo modo, *StoriaGuida* fornisce evidenze su *AvversioneRischio* e *CapacitàGuida*, che a loro volta aiutano a prevedere il futuro *ComportamentoGuida*.

ComportamentoGuida può essere considerata una tendenza a guidare in un modo che favorisce gli incidenti, misurata per chilometro. Il fatto che un *Incidente* si verifichi effettivamente in un periodo di tempo fissato dipende anche dal *Chilometraggio* annuale e dalle *FunzioniSicurezza* del veicolo. Se si verifica un *Incidente*, ci sono tre tipi di costi: il *CostoMedico*

⁴ Alcune compagnie assicuratrici acquisiscono anche la storia di credito del richiedente e la utilizzano per valutare il rischio; questa infatti fornisce più informazioni riguardo la categoria socioeconomica. Quando si utilizzano variabili nascoste di questo tipo, occorre prestare attenzione che non diventino inavvertitamente proxy per variabili quali la razza che non possono essere usate nelle decisioni assicurative. Alcune tecniche per evitare bias di questo tipo sono descritte nel Capitolo 19 del Volume 2.

per il richiedente dipende da *Età* e da *Assorbimento* (dell'auto), che dipende a sua volta dalla *Robustezza* dell'auto e dal fatto che abbia o meno un *Airbag*; il *CostoResponsabilità* (cure mediche, dolore e sofferenza, perdita di reddito, ecc.) per l'altro guidatore; e il *CostoPossesso* per il richiedente e l'altro guidatore, che dipendono entrambi (in modi diversi) dalla *Robustezza* dell'auto e dal *ValoreAuto* per il richiedente.

Abbiamo spiegato il tipo di ragionamento necessario per sviluppare la topologia e le variabili nascoste in una rete bayesiana; dobbiamo anche specificare gli intervalli e le distribuzioni condizionate per ciascuna variabile. Per quanto riguarda gli intervalli, la principale decisione è spesso quella tra definire la variabile discreta o continua. Per esempio, la *Robustezza* del veicolo potrebbe essere una variabile continua compresa tra 0 e 1, oppure una variabile discreta con intervallo {*Lattina*, *Normale*, *Carroarmato*}.

Le variabili continue offrono una precisione maggiore, ma rendono impossibile l'inferenza esatta, tranne in pochi casi particolare. Nel caso di una variabile discreta con molti valori possibili potrebbe diventare noioso compilare le tabelle delle probabilità condizionate, che sarebbero molto grandi, e l'inferenza esatta diventerebbe più costosa, a meno che il valore della variabile non sia sempre osservato. Per esempio, *ModelloMarca* in un sistema reale avrebbe migliaia di valori possibili, per cui la sua variabile figlia *ValoreAuto* avrebbe una tabella CPT enorme che dovrebbe essere compilata prendendo i dati da database industriali; tuttavia, poiché il *ModelloMarca* è sempre osservato, ciò non aumenta la complessità dell'inferenza: in effetti, i valori osservati per i tre genitori individuano esattamente una riga della tabella CPT per *ValoreAuto*.

Le distribuzioni condizionate per questo modello sono fornite nel repository di codice per il libro; è inclusa una versione con sole variabili discrete, per cui è possibile eseguire inferenze esatte. Nella realtà, molte variabili sarebbero continue e le distribuzioni condizionate dovrebbero essere apprese dai dati storici sui richiedenti e le loro storie assicurative. Vedremo come apprendere modelli di reti bayesiane nel Capitolo 20 del Volume 2.

Ora affronteremo la domanda finale: come effettuare inferenze nella rete per formulare previsioni. Ogni metodo di inferenza che descriveremo verrà valutato sulla rete delle assicurazioni per misurarne i requisiti temporali e spaziali.

13.3 Inferenza esatta nelle reti bayesiane

evento

Il compito principale di ogni sistema di inferenza probabilistica è calcolare la distribuzione di probabilità a posteriori di un insieme di **variabili di query**, dato qualche **evento** osservato, termine con cui solitamente si intende un assegnamento di valori a un insieme di **variabili di evidenza**.⁵ Per semplificare la trattazione, considereremo una sola variabile di query per volta; l'algoritmo può essere facilmente esteso a query con più variabili; per esempio, possiamo risolvere la query $\mathbf{P}(U,V|\mathbf{e})$ moltiplicando $\mathbf{P}(V|\mathbf{e})$ e $\mathbf{P}(U|V,\mathbf{e})$. Useremo la notazione introdotta nel Capitolo 12: X indica la variabile di query; \mathbf{E} l'insieme di variabili di evidenza E_1, \dots, E_m ; \mathbf{e} un particolare evento osservato; \mathbf{Y} le variabili nascoste Y_1, \dots, Y_l (talvolta chiamate anche variabili non query e non di evidenza). Così, l'insieme completo di variabili è $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$. Tipicamente, un'interrogazione riguarderà una distribuzione di probabilità a posteriori $\mathbf{P}(X|\mathbf{e})$.

Nella rete dell'antifurto, potremmo osservare l'evento in cui *JohnTelefona = true* e *MaryTelefona = true*. Potremmo allora domandarci qual è la probabilità che si sia verificata un'intrusione:

$$\mathbf{P}(\text{Intrusione} | \text{JohnTelefona} = \text{true}, \text{MaryTelefona} = \text{true}) = \langle 0,284, 0,716 \rangle .$$

⁵ Un'altra attività ampiamente studiata è quella di trovare la **spiegazione più probabile** per un'evidenza osservata. Questa e altre attività sono discusse nelle note storiche e bibliografiche a fine capitolo.

In questo paragrafo discuteremo alcuni algoritmi esatti per il calcolo delle probabilità a posteriori ed esamineremo la complessità di questo compito. Risulterà che il caso generale è intrattabile, ragion per cui il Paragrafo 13.4 presenterà metodi di inferenza approssimata.

13.3.1 Inferenza per enumerazione

Nel Capitolo 12 abbiamo spiegato che ogni probabilità condizionata può essere calcolata semplicemente sommando i termini relativi della distribuzione congiunta completa. Specificatamente, a una query $\mathbf{P}(X|\mathbf{e})$ si può rispondere applicando l'Equazione (12.9), che riportiamo per comodità:

$$\mathbf{P}(X|\mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}).$$

Ora, una rete bayesiana fornisce una rappresentazione della distribuzione congiunta completa; più precisamente, l'Equazione (13.2) mostra che i termini $P(x, \mathbf{e}, \mathbf{y})$ della distribuzione congiunta si possono scrivere come prodotti di probabilità condizionate prese dalla rete. Quindi *si può dare risposta a un'interrogazione attraverso una rete bayesiana, sommando i prodotti delle probabilità condizionate prese dalla rete stessa*.



Considerate la query $\mathbf{P}(\text{Intrusione}|\text{JohnTelefona} = \text{true}, \text{MaryTelefona} = \text{true})$. Le variabili nascoste sono *Terremoto* e *Allarme*. Dall'Equazione (12.9), usando le lettere iniziali delle variabili per abbreviare le espressioni, risulta

$$\mathbf{P}(I|j,m) = \alpha \mathbf{P}(I,j,m) = \alpha \sum_e \sum_a \mathbf{P}(I,j,m,e,a).$$

La semantica delle reti bayesiane (Equazione (13.2)) ci fornisce quindi un'espressione in termini di elementi di CPT. Per semplicità, considereremo solo il caso in cui *Intrusione = true*:

$$P(i|j,m) = \alpha \sum_e \sum_a P(i) P(t) P(a|i,t) P(j|a) P(m|a). \quad (13.4)$$

Per calcolare questa espressione occorre sommare quattro termini, ognuno dei quali è ottenuto moltiplicando cinque numeri. Nel caso pessimo, in cui dobbiamo sommare quasi tutte le variabili, ci saranno $O(2^n)$ termini nella somma, ognuno dei quali è un prodotto di $O(n)$ valori di probabilità, per cui un'implementazione ingenua avrebbe complessità $O(n2^n)$.

È possibile ridurre la complessità a $O(2^n)$ sfruttando la struttura annidata del calcolo. In termini simbolici, ciò significa spostare le sommatorie più all'interno possibile nelle espressioni come l'Equazione (13.4). Possiamo farlo perché non tutti i fattori del prodotto di probabilità dipendono da tutte le variabili. Abbiamo quindi:

$$P(i|j,m) = \alpha P(i) \sum_e P(t) \sum_a P(a|i,t) P(j|a) P(m|a). \quad (13.5)$$

Questa espressione può essere valutata iterando su tutte le variabili in ordine e moltiplicando man mano i valori delle CPT. Per ogni somma è necessario anche iterare su tutti i possibili valori della variabile. La struttura di questo calcolo è mostrata in forma di albero nella Figura 13.10. Usando i valori della Figura 13.2, otteniamo che $P(i|j, m) = \alpha \times 0,00059224$. Il calcolo relativo a $\neg i$ dà come risultato $\alpha \times 0,0014919$; quindi

$$\mathbf{P}(I|j, m) = \alpha \langle 0,00059224, 0,0014919 \rangle \approx \langle 0,284, 0,716 \rangle.$$

In definitiva la probabilità che si sia verificata un'intrusione, data la chiamata di entrambi i vicini, è pari a circa il 28%.

L'algoritmo ENUMERAZIONE-ASK della Figura 13.11 valuta questi alberi mediante una ricorsione in profondità, da sinistra a destra. La struttura dell'algoritmo è molto simile a quella dell'algoritmo con backtracking per la risoluzione di CSP (Figura 6.5) e all'algoritmo

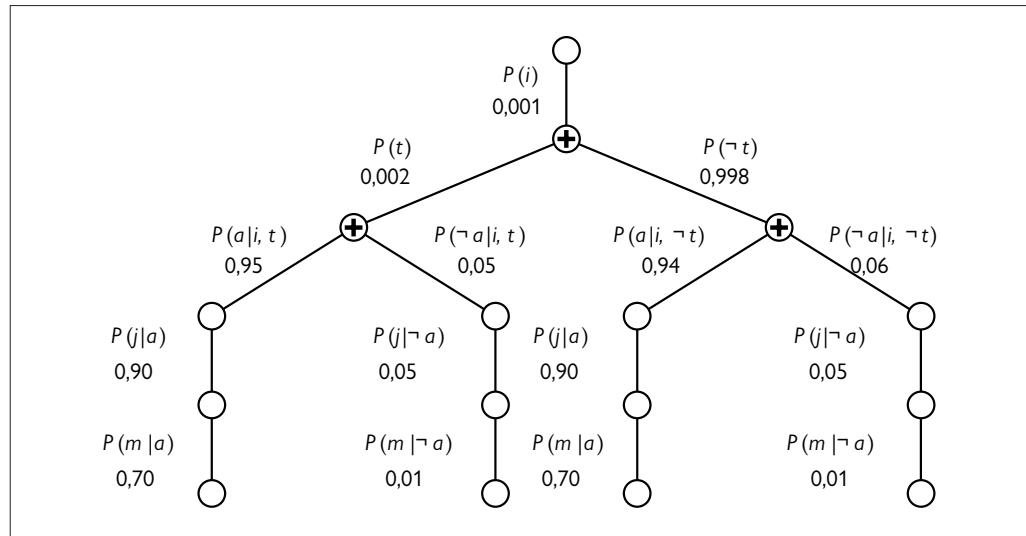


Figura 13.10 La struttura dell'espressione dell'Equazione (13.5). Il processo di valutazione procede dall'alto verso il basso, moltiplicando i valori lungo ogni cammino e sommando nei nodi “+”. Notate la ripetizione dei cammini per j e m .

Figura 13.11

L'algoritmo di enumerazione per inferenza esatta in reti bayesiane.

```

function ENUMERAZIONE-ASK( $X, \mathbf{e}, bn$ ) returns una distribuzione su  $X$ 
  inputs:  $X$ , la variabile di query
   $\mathbf{e}$ , i valori osservati delle variabili  $\mathbf{E}$ 
   $bn$ , una rete bayesiana con variabili variabili

   $\mathbf{Q}(X) \leftarrow$  una distribuzione su  $X$ , inizialmente vuota
  for each valore  $x_i$  di  $X$  do
     $\mathbf{Q}(x_i) \leftarrow$  ENUMERA-TUTTE(variabili,  $\mathbf{e}_{xi}$ )
    dove  $\mathbf{e}_{xi}$  è  $\mathbf{e}$  estesa con  $X = x_i$ 
  return NORMALIZZA( $\mathbf{Q}(X)$ )

function ENUMERA-TUTTE(variabili,  $\mathbf{e}$ ) returns un numero reale
  if VUOTO?(variabili) then return 1,0
   $V \leftarrow$  PRIMO(variabili)
  if  $V$  è una variabile di evidenza con valore  $v$  in  $\mathbf{e}$ 
    then return  $P(v|genitori(V)) \times$  ENUMERA-TUTTE(RESTO(variabili),  $\mathbf{e}$ )
    else return  $\sum_v P(v|genitori(V)) \times$  ENUMERA-TUTTE(RESTO(variabili),  $\mathbf{e}_v$ )
    dove  $\mathbf{e}_v$  corrisponde a  $\mathbf{e}$  estesa con  $V = v$ 

```

DPLL per la soddisfacibilità (Figura 7.17). La sua complessità spaziale è solo lineare nel numero delle variabili: l'algoritmo somma valori presi da tutta la distribuzione congiunta completa senza mai costruirla esplicitamente. Sfortunatamente, la sua complessità temporale per una rete con n variabili booleane (senza contare le variabili di evidenza) è sempre $O(2^n)$: meglio dell' $O(n2^n)$ del semplice approccio descritto in precedenza, ma comunque

piuttosto difficile da gestire. Per quanto riguarda la rete dell'esempio sulle assicurazioni illustrata nella Figura 13.9, che è relativamente piccola, l'inferenza esatta usando l'enumerazione richiede circa 227 milioni di operazioni aritmetiche per una tipica interrogazione sulle variabili di costo.

Se osservate attentamente l'albero della Figura 13.10, tuttavia, noterete che contiene *sot-toespressioni ripetute* valutate. I prodotti $P(j|a)P(m|a)$ e $P(j|\neg a)P(m|\neg a)$ sono calcolati due volte, una per ogni valore di E . La chiave per un'inferenza efficiente nelle reti bayesiane sta nell'evitare tali calcoli sprecati, e nel prossimo paragrafo descriveremo un metodo generale che consente di farlo.

13.3.2 L'algoritmo di eliminazione delle variabili

L'algoritmo di enumerazione può essere migliorato sensibilmente eliminando calcoli ripetuti del tipo mostrato nella Figura 13.10. L'idea è semplice: una volta svolto il calcolo, il risultato è memorizzato per uso futuro. Questa è una forma di programmazione dinamica. Ci sono molte versioni di quest'approccio; noi presentiamo il più semplice, l'algoritmo di **eliminazione delle variabili**. L'algoritmo opera valutando espressioni come l'Equazione (13.5) *da destra a sinistra* (ovvero, nella Figura 13.10, *dal basso verso l'alto*). I risultati intermedi sono memorizzati, e le somme su ogni variabile sono svolte solo per le porzioni dell'espressione che dipendono da quella variabile.

Esemplifichiamo il processo ricorrendo ancora alla rete dell'antifurto: valutiamo l'espressione

$$\mathbf{P}(I|j,m) = \alpha \underbrace{\mathbf{P}(I)}_{\mathbf{f}_1(I)} \sum_a \underbrace{\mathbf{P}(t)}_{\mathbf{f}_2(T)} \sum_a \underbrace{\mathbf{P}(a|I,t)}_{\mathbf{f}_3(A,I,T)} \underbrace{P(j|a)}_{\mathbf{f}_4(A)} \underbrace{P(m|a)}_{\mathbf{f}_5(A)}.$$

Notate che abbiamo indicato, per ogni parte dell'espressione, il nome del corrispondente **fattore**; ogni fattore è una matrice indicizzata con i valori delle variabili che compaiono come suoi argomenti. Per esempio, i fattori $\mathbf{f}_4(A)$ e $\mathbf{f}_5(A)$ corrispondenti a $P(j|a)$ e $P(m|a)$ dipendono soltanto da A perché J e M sono fissate dalla query. Sono, quindi, vettori di due elementi:

$$\mathbf{f}_4(A) = \begin{pmatrix} P(j|a) \\ P(j|\neg a) \end{pmatrix} = \begin{pmatrix} 0,90 \\ 0,05 \end{pmatrix} \quad \mathbf{f}_5(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix} = \begin{pmatrix} 0,70 \\ 0,01 \end{pmatrix}.$$

$\mathbf{f}_3(A, I, T)$ sarà una matrice $2 \times 2 \times 2$, difficile da mostrare sulla carta (il "primo" elemento è dato da $P(a|i, t) = 0,95$ mentre l'"ultimo" è dato da $P(\neg a|\neg i, \neg t) = 0,999$.) In termini di fattori, l'espressione dell'interrogazione è scritta come:

$$\mathbf{P}(I|j,m) = \alpha \mathbf{f}_1(I) \times \sum_t \mathbf{f}_2(T) \times \sum_a \mathbf{f}_3(A, I, T) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A).$$

Dove in questo caso l'operatore " \times " non indica la normale moltiplicazione di matrici, ma l'operazione di **prodotto puntuale**, che descriveremo tra breve.

Il processo di valutazione somma le variabili (da destra a sinistra) dai prodotti puntuali di fattori per produrre nuovi fattori, e alla fine si ottiene un fattore che costituisce la soluzione, ovvero la distribuzione a posteriori sulla variabile query. I passaggi sono i seguenti.

- Per prima cosa eliminiamo A dal prodotto di \mathbf{f}_3 , \mathbf{f}_4 e \mathbf{f}_5 con un *summing out*. Otteniamo così un nuovo fattore $2 \times 2 \mathbf{f}_6(I, T)$ i cui indici variano soltanto su I ed T :

$$\begin{aligned} \mathbf{f}_6(I, T) &= \sum_a \mathbf{f}_3(A, I, T) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) \\ &= (\mathbf{f}_3(a, I, T) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a)) + (\mathbf{f}_3(\neg a, I, T) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a)). \end{aligned}$$

Ora rimane l'espressione:

$$\mathbf{P}(I | j, m) = \alpha \mathbf{f}_1(I) \times \sum_t \mathbf{f}_2(T) \times \mathbf{f}_6(I, T).$$

- Poi eliminiamo T dal prodotto di \mathbf{f}_2 e \mathbf{f}_6 con un summing out:

$$\begin{aligned}\mathbf{f}_7(I) &= \sum_t \mathbf{f}_2(T) \times \mathbf{f}_6(I, T) \\ &= \mathbf{f}_2(t) \times \mathbf{f}_6(I, t) + \mathbf{f}_2(\neg t) \times \mathbf{f}_6(I, \neg t).\end{aligned}$$

Rimane così l'espressione:

$$\mathbf{P}(I | j, m) = \alpha \mathbf{f}_1(I) \times \mathbf{f}_7(I)$$

che può essere valutata effettuando il prodotto puntuale e poi normalizzando il risultato.

Esaminando questa sequenza di passi, vediamo che le operazioni base richieste sono due: il prodotto puntuale di una coppia di fattori e il summing out, ovvero l'eliminazione, attraverso la somma, di una variabile da un prodotto di fattori. Queste operazioni sono descritte nel paragrafo seguente.

Operazioni sui fattori

Il prodotto puntuale di due fattori \mathbf{f} e \mathbf{g} fornisce un nuovo fattore \mathbf{h} le cui variabili sono l'*unione* delle variabili in \mathbf{f} e \mathbf{g} e i cui elementi sono dati dal prodotto degli elementi corrispondenti nei due fattori. Supponiamo che i due fattori abbiano le variabili Y_1, \dots, Y_k in comune. Risulta allora:

$$\mathbf{f}(X_1 \dots X_j, Y_1 \dots Y_k) \times \mathbf{g}(Y_1 \dots Y_k, Z_1 \dots Z_\ell) = \mathbf{h}(X_1 \dots X_j, Y_1 \dots Y_k, Z_1 \dots Z_\ell).$$

Se tutte le variabili sono binarie, \mathbf{f} e \mathbf{g} hanno rispettivamente 2^{j+k} e $2^{k+\ell}$ elementi, e il prodotto puntuale ne ha $2^{j+k+\ell}$. Per esempio, dati due fattori $\mathbf{f}(X, Y)$ e $\mathbf{g}(Y, Z)$, il prodotto puntuale $\mathbf{f} \times \mathbf{g} = \mathbf{h}(X, Y, Z)$ ha $2^{1+1+1} = 8$ elementi, come illustrato nella Figura 13.12.

Notate che il fattore risultante da un prodotto puntuale può contenere più variabili di ognuno dei fattori moltiplicato, e che la dimensione di un fattore è esponenziale nel numero di variabili. È proprio qui che si annida la complessità spaziale e temporale nell'algoritmo di eliminazione di variabili.

X	Y	f(X, Y)	Y	Z	g(Y, Z)	X	Y	Z	h(X, Y, Z)
t	t	0,3	t	t	0,2	t	t	t	0,3 × 0,2
t	f	0,7	t	f	0,8	t	t	f	0,3 × 0,8
f	t	0,9	f	t	0,6	t	f	t	0,7 × 0,6
f	f	0,1	f	f	0,4	t	f	f	0,7 × 0,4
						f	t	t	0,9 × 0,2
						f	t	f	0,9 × 0,8
						f	f	t	0,1 × 0,6
						f	f	f	0,1 × 0,4

Figura 13.12 Moltiplicazione puntuale: $\mathbf{f}(X, Y) \times \mathbf{g}(Y, Z) = \mathbf{h}(X, Y, Z)$.

function ELIMINAZIONE-ASK(X, \mathbf{e}, bn) **returns** una distribuzione su X

inputs: X , la variabile di query
 \mathbf{e} , valori osservati per variabili E
 bn , una rete bayesiana con variabili *variabili*

```

fattori  $\leftarrow [ ]$ ;
for each  $V$  in ORDINA(variabili) do
    fattori  $\leftarrow [\text{CREA-FATTORE}(V, \mathbf{e})] + fattori$ 
    if  $V$  è una variabile nascosta then fattori  $\leftarrow \text{SUM-OUT}(V, fattori)$ 
return NORMALIZZA(PRODOTTO-PUNTUALE(fattori))

```

Figura 13.13
L'algoritmo di eliminazione delle variabili per l'inferenza esatta in reti bayesiane.

Il *summing out* di una variabile da un prodotto di fattori si effettua sommando le sottomatrici formate fissando la variabile a ognuno dei suoi valori. Per esempio, per eliminare X da $\mathbf{h}(X, Y, Z)$ mediante summing out scriviamo:

$$\begin{aligned} \mathbf{h}_2(Y, Z) &= \sum_x \mathbf{h}(X, Y, Z) = \mathbf{h}(x, Y, Z) + \mathbf{h}(\neg x, Y, Z) \\ &= \begin{pmatrix} 0,06 & 0,24 \\ 0,42 & 0,28 \end{pmatrix} + \begin{pmatrix} 0,18 & 0,72 \\ 0,06 & 0,04 \end{pmatrix} = \begin{pmatrix} 0,24 & 0,96 \\ 0,48 & 0,32 \end{pmatrix}. \end{aligned}$$

L'unico trucco è quello di notare che ogni fattore che *non* dipende dalla variabile da eliminare con il summing out può essere portato fuori dalla sommatoria. Per esempio, per eliminare X dal prodotto di \mathbf{f} e \mathbf{g} , possiamo spostare \mathbf{g} al di fuori della sommatoria:

$$\sum_x \mathbf{f}(X, Y) \times \mathbf{g}(Y, Z) = \mathbf{g}(Y, Z) \times \sum_x \mathbf{f}(X, Y).$$

Questo metodo è, potenzialmente, molto più efficiente rispetto a calcolare prima il prodotto puntuale \mathbf{h} , più grande, e poi eliminare X con il summing out.

Noteate che le matrici *non* vengono moltiplicate finché non dobbiamo eliminare una variabile dal prodotto accumulato. A quel punto, moltiplichiamo solo le matrici che includono la variabile da eliminare. Date le funzioni per il prodotto puntuale e per il summing out, l'algoritmo di eliminazione delle variabili è abbastanza semplice da scrivere, come si vede nella Figura 13.13.

Ordinamento delle variabili e rilevanza delle variabili

L'algoritmo della Figura 13.13 include una funzione ORDINA, non specificata, per scegliere un ordinamento delle variabili. Ogni scelta porta a un algoritmo valido, ma ordinamenti diversi possono causare la generazione di fattori intermedi diversi durante il calcolo. Per esempio, nel calcolo mostrato in precedenza abbiamo eliminato A prima T ; se procediamo nell'altro ordine, il calcolo diventa

$$\mathbf{P}(I | j, m) = \alpha \mathbf{f}_1(I) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_t \mathbf{f}_2(T) \times \mathbf{f}_3(A, I, T),$$

che porta alla generazione di un nuovo fattore $\mathbf{f}_6(A, I)$.

In generale, i requisiti temporali e spaziali dell'eliminazione di variabili sono dominati dalla dimensione del fattore più grande costruito durante l'esecuzione dell'algoritmo. Questo a sua volta è determinato dall'ordine di eliminazione delle variabili e dalla struttura della rete. Determinare l'ordinamento ottimo è un problema intrattabile, ma sono disponibili buo-

ne euristiche. Un metodo abbastanza efficace è quello greedy: eliminare ogni variabile che minimizza la dimensione del prossimo fattore che verrà costruito.

Consideriamo ancora una query: $P(JohnTelefona|Intrusione = true)$. Come al solito (cfr. Equazione(13.5)), il primo passo è scrivere la sommatoria annidata:

$$\mathbf{P}(J | i) = \alpha P(i) \sum_t P(t) \sum_a P(a | i, t) \mathbf{P}(J | a) \sum_m P(m | a).$$

Valutando quest'espressione da destra a sinistra, notiamo un punto interessante: $\sum_m P(m | a)$ è uguale a 1 per definizione! Quindi non c'è alcun bisogno di includerla; la variabile M è *irrilevante* per quest'interrogazione. Un altro modo di enunciare la stessa cosa è dire che il risultato della query $P(JohnTelefona|Intrusione = true)$ non cambia se si elimina completamente *MaryTelefona* dalla rete. In generale possiamo rimuovere qualsiasi nodo foglia che non è una variabile di query né di evidenza. La sua rimozione potrebbe creare nuovi nodi foglia, e anche questi potrebbero essere irrilevanti. Continuando questo processo, si arriva infine ad appurare che *ogni variabile che non è un progenitore di una variabile di query o di evidenza è irrilevante per l'interrogazione*. Un algoritmo di eliminazione di variabili può quindi eliminare tutte queste variabili ancora prima di valutare la query.

Se si applica l'eliminazione di variabili alla rete per l'esempio sulle assicurazioni della Figura 13.9, si ottiene un notevole miglioramento rispetto al semplice algoritmo di enumerazione. Usando l'ordinamento topologico inverso per le variabili, l'inferenza esatta con l'eliminazione è circa mille volte più veloce dell'algoritmo di enumerazione.



13.3.3 La complessità dell'inferenza esatta

singolarmente connesse polialberi



connessioni multiple



riduzione

La complessità dell'inferenza esatta in reti bayesiane dipende fortemente dalla struttura della rete. La rete dell'antifurto della Figura 13.2 appartiene a una famiglia di reti in cui tra due nodi qualsiasi c'è al più un cammino non orientato (cioè che ignora la direzione delle frecce). Queste reti sono chiamate **singolarmente connesse** o **polialberi**, e hanno una proprietà particolarmente gradevole: *la complessità temporale e spaziale dell'inferenza esatta nei polialberi è lineare nelle dimensioni della rete*. Qui le dimensioni sono definite dal numero degli elementi delle tabelle CPT; se il numero di genitori è limitato da una costante, anche la complessità sarà lineare nel numero di nodi. Questi risultati valgono per qualsiasi ordinamento consistente con quello topologico della rete (cfr. Esercizio 13.VEEX).

Per reti a **connessioni multiple**, come quella dell'esempio sulle assicurazioni mostrata nella Figura 13.9, la complessità spaziale e temporale dell'eliminazione delle variabili può, nel caso pessimo, crescere esponenzialmente anche quando il numero di genitori per nodo è limitato. Questo non sorprende quando si considera che, *dato che include come caso speciale l'inferenza logica proposizionale, l'inferenza nelle reti bayesiane è NP-difficile*.

Per dimostrarlo dobbiamo determinare come codificare un problema di soddisfacibilità proposizionale in una rete bayesiana, in modo che effettuando un'inferenza su questa rete possiamo sapere se le formule proposizionali originali sono soddisfacibili oppure no (nel linguaggio della teoria della complessità si dice che **riduciamo** i problemi di soddisfacibilità a problemi di inferenza in reti bayesiane). La procedura è abbastanza semplice. La Figura 13.14 mostra come codificare un particolare problema 3-SAT. Le variabili proposizionali diventano le variabili radice della rete, ognuna con probabilità a priori di 0,5. Lo strato di nodi successivo corrisponde alle clausole, dove ogni variabile clausola C_j è connessa alle variabili appropriate come genitori. La distribuzione condizionata per una variabile clausola è una disgiunzione deterministica, con negazione ove necessario, in modo che ogni variabile clausola è vera se e solo se l'assegnamento ai suoi genitori soddisfa tale clausola. Infine, S è la congiunzione delle variabili clausola.

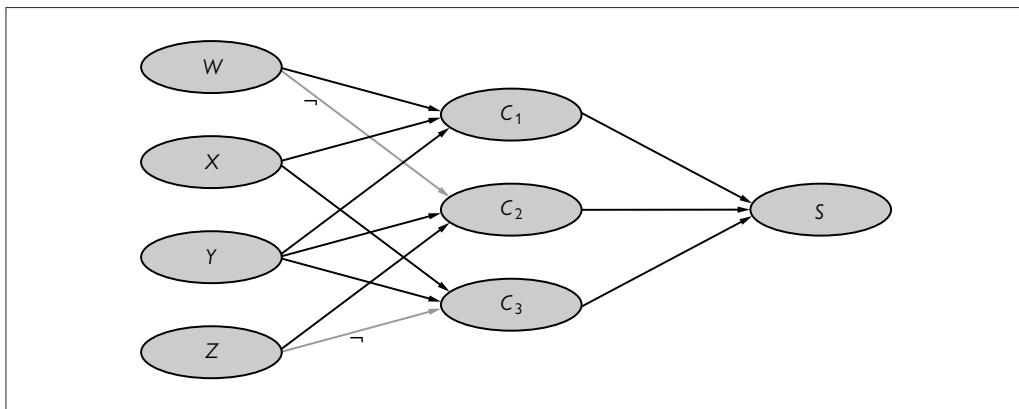


Figura 13.14 Codifica in rete bayesiana della formula 3-CNF
 $(W \vee X \vee Y) \wedge (\neg W \vee Y \vee Z) \wedge (X \vee Y \vee \neg Z)$.

Per determinare se la formula originale è soddisfacibile, valutiamo $P(S = \text{true})$. Se la formula è *soddisfacibile*, allora esiste qualche possibile assegnamento alle variabili logiche che rende S vera; nella rete bayesiana, questo significa che esiste un mondo possibile con probabilità non nulla in cui le variabili radice hanno tale assegnamento, le variabili clausola hanno valore *true* e S ha valore *true*. Perciò, $P(S = \text{true}) > 0$ per una formula soddisfacibile. Viceversa, $P(S = \text{true}) = 0$ per una formula non soddisfacibile: tutti i mondi con $S = \text{true}$ hanno probabilità 0. Possiamo quindi usare l'inferenza in reti bayesiane per risolvere problemi 3-SAT; da ciò concludiamo che l'inferenza in reti bayesiane è NP-difficile.

Ma possiamo fare anche di più. Notate che la probabilità di ogni assegnamento soddisfacente è 2^{-n} per un problema con n variabili. Quindi, il *numero* di assegnamenti soddisfacenti è $P(S = \text{true})/(2^{-n})$. Poiché il calcolo del *numero* di assegnamenti soddisfacenti per un problema 3-SAT è #P-completo (“numero-P completo” o “sharp-P completo”), questo significa che l'inferenza in reti bayesiane è #P-difficile, cioè che è strettamente più difficile dei problemi NP-completi.

Esiste una stretta connessione tra la complessità dell'inferenza in una rete bayesiana e quella dei problemi di soddisfacimento di vincoli (CSP). Come abbiamo visto nel Capitolo 6, la difficoltà della soluzione di un CSP discreto è legata a quanto il suo grafo dei vincoli “assomiglia a un albero”. Misure come la **larghezza d'albero**, che limitano la complessità della risoluzione di un CSP, possono anche essere applicate direttamente alle reti bayesiane. Inoltre, l'algoritmo di eliminazione delle variabili può essere generalizzato per risolvere anche i CSP.

Come possiamo ridurre i problemi di soddisfacibilità a inferenza in reti bayesiane, possiamo ridurre l'inferenza in reti bayesiane alla soddisfacibilità, il che ci consente di sfruttare i potenti meccanismi sviluppati per la risoluzione di problemi SAT (cfr. Capitolo 7). In questo caso, la riduzione è una particolare forma di risoluzione SAT denominata **conteggio con modello ponderato** (WMC, *weighted model counting*). Mentre nel conteggio con modello normale si conta il numero di assegnamenti soddisfacenti per un'espressione SAT, nel conteggio WMC si sommano i pesi totali di tali assegnamenti soddisfacenti, dove in questo caso il peso è sostanzialmente il prodotto delle probabilità condizionate per ciascun assegnamento di variabile dati i suoi genitori (cfr. l'Esercizio 13.WMCX per i dettagli). Grazie anche al fatto che la tecnologia di risoluzione SAT è stata così bene ottimizzata per applicazioni su larga scala, l'inferenza in reti bayesiane via WMC è competitiva e talvolta superiore ad altri algoritmi esatti su reti con ampia larghezza d'albero.

conteggio con modello ponderato

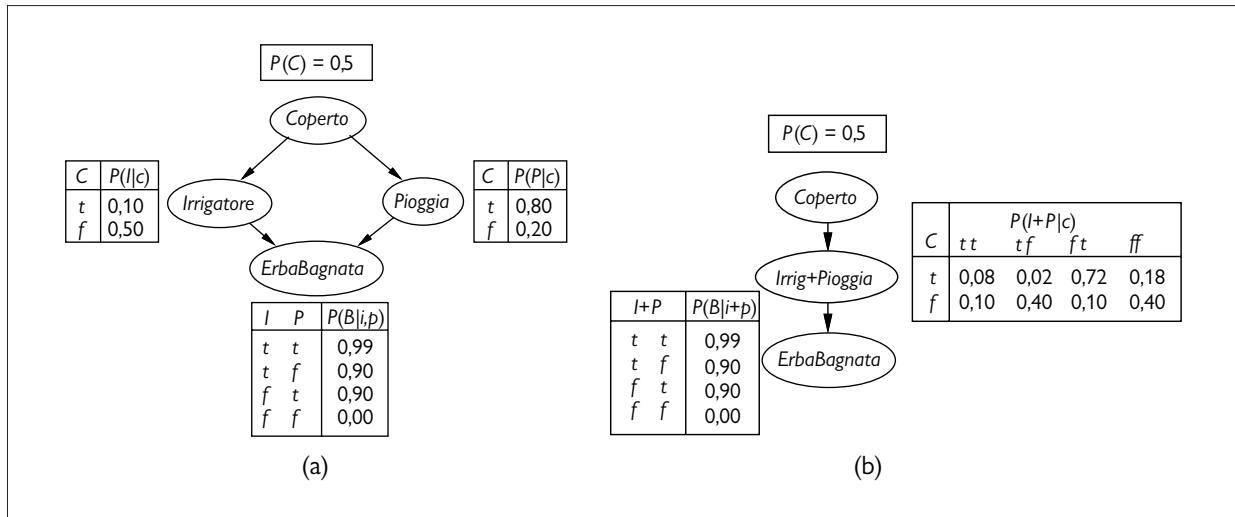


Figura 13.15 (a) Una rete a connessioni multiple che descrive la routine quotidiana con cui Mary si occupa del prato: ogni mattina controlla le condizioni meteorologiche: se è coperto, solitamente non attiva l’irrigatore; se l’irrigatore è attivo, o se durante il giorno piove, l’erba sarà bagnata. Quindi, *Coperto* influisce su *ErbaBagnata* attraverso due diversi percorsi causali. (b) Una rete equivalente alla rete a connessioni multiple, ottenuta mediante il clustering (raggruppamento).

13.3.4 Algoritmi di clustering

clustering
albero di join

meganodo

L’algoritmo di eliminazione delle variabili è semplice ed efficiente quando si deve dare una risposta a interrogazioni singole. Se vogliamo calcolare la probabilità a posteriori di tutte le variabili in una rete, tuttavia, può risultare meno efficiente. Per esempio, in una rete polialbero sarebbe necessario porre $O(n)$ query separate, ognuna di costo $O(n)$, per un tempo totale $O(n^2)$. Con gli algoritmi di **clustering** o di **raggruppamento**, noti anche come algoritmi ad **albero di join**, il tempo può essere ridotto a $O(n)$. Per questa ragione tali algoritmi sono ampiamente usati negli strumenti commerciali per l’elaborazione di reti bayesiane.

L’idea base del clustering è raggruppare singoli nodi della rete in “nodi cluster” aggregati in modo tale che la rete risultante sia un polialbero. Per esempio, la rete a connessioni multiple della Figura 13.15(a) può essere trasformata in polialbero unendo i nodi *Irrigatore* e *Pioggia* in un cluster chiamato *Irrigatore+Pioggia*, come si vede nella Figura 13.15(b). I due nodi booleani sono rimpiazzati da un **meganodo** che ha quattro possibili valori: *tt*, *tf*, *ft* e *ff*. Il meganodo ha un solo genitore, la variabile booleana *Coperto*, per cui i casi condizionanti sono due. Anche se in questo esempio non lo si vede, il processo di clustering spesso produce meganodi che condividono alcune variabili.

Una volta che la rete ha assunto la forma di un polialbero, è richiesto un algoritmo di inferenza specializzato, perché i metodi di inferenza ordinari non sono in grado di gestire meganodi che condividono variabili tra di loro. In sostanza l’algoritmo è una forma di propagazione di vincoli (cfr. Capitolo 6) dove i vincoli assicurano che meganodi adiacenti concordino sulle probabilità a posteriori di tutte le variabili in comune. Con una gestione accurata delle informazioni parziali quest’algoritmo può calcolare le probabilità a posteriori di tutti i nodi non di evidenza della rete in un tempo *lineare* nella dimensione della rete ottenuta con il clustering. Naturalmente, la NP-difficoltà del problema non è scomparsa: se una rete richiede un tempo e uno spazio esponenziali con l’eliminazione delle variabili, le CPT della rete con nodi raggruppati saranno esponenzialmente grandi.

13.4 Inferenza approssimata nelle reti bayesiane

Vista l'intrattabilità dell'inferenza esatta in reti di grandi dimensioni, prendiamo in considerazione l'uso di metodi approssimati. Questo paragrafo descrive tecniche di campionamento randomizzato, chiamati anche algoritmi **Monte Carlo**, che forniscono risposte approssimate la cui accuratezza dipende dal numero di campioni generati. Questi algoritmi operano generando eventi casuali basati sulle probabilità della rete bayesiana e contando le diverse risposte trovate in tali eventi. Con un numero di campioni sufficiente possiamo arrivare arbitrariamente vicino a recuperare la distribuzione di probabilità reale, purché la rete bayesiana non abbia distribuzioni condizionate deterministiche.

Monte Carlo

Gli algoritmi Monte Carlo, tra cui rientra il simulated annealing (cfr. Paragrafo 4.1.2 del Capitolo 4) sono usati in molti ambiti della scienza per stimare quantità difficili da calcolare esattamente. In questo paragrafo ci interessa applicare il campionamento al calcolo della probabilità a posteriori nelle reti bayesiane. Descriveremo due famiglie di algoritmi: il campionamento diretto e quello con catene di Markov. Diversi altri approcci per l'inferenza approssimata sono menzionati nelle note alla fine di questo capitolo.

13.4.1 Metodi di campionamento diretto

L'elemento principale di ogni algoritmo di campionamento è la generazione di campioni a partire da una distribuzione di probabilità nota. Per esempio, una moneta ben bilanciata può essere considerata una variabile casuale *Moneta* con valori *{testa, croce}* e una distribuzione a priori $\mathbf{P}(\text{Moneta}) = \langle 0,5, 0,5 \rangle$. Campionare questa distribuzione è esattamente come lanciare la moneta: uscirà *testa* oppure *croce* con la stessa probabilità di 0,5. Data una sorgente di numeri casuali *r* uniformemente distribuiti nell'intervallo [0, 1], è molto semplice campionare una qualsiasi distribuzione su una singola variabile, discreta o continua che sia: si costruisce la distribuzione cumulata per la variabile e si restituisce il primo valore la cui probabilità cumulata è superiore a *r* (cfr. Esercizio 13.PRSA).

Iniziamo con un processo di campionamento casuale per una rete bayesiana che non ha associata alcuna evidenza. L'idea è di campionare a turno ogni variabile, in ordine topologico. La distribuzione di probabilità da cui si campiona il valore è condizionata ai valori già assegnati ai genitori della variabile (poiché campioniamo in ordine topologico, i genitori hanno sicuramente dei valori). Questo algoritmo è mostrato nella Figura 13.16; applicandolo alla rete della Figura 13.15(a) con l'ordinamento *Coperto, Irrigatore, Pioggia, ErbaBagnata* potremmo produrre un evento casuale come segue:

1. Campione da $\mathbf{P}(\text{Coperto}) = \langle 0,5, 0,5 \rangle$; il valore è *true*.
2. Campione da $\mathbf{P}(\text{Irrigatore}|\text{Coperto} = \text{true}) = \langle 0,1, 0,9 \rangle$; il valore è *false*.

```

function CAMPIONA-PRIORI(bn) returns
    un evento campionato partendo dalla distribuzione a priori specificata da bn
inputs: bn, una rete bayesiana che specifica la distribuzione congiunta  $\mathbf{P}(X_1, \dots, X_n)$ 

    x  $\leftarrow$  un evento con n elementi
    for each variabile  $X_i$  in  $X_1, \dots, X_n$  do
        x[i]  $\leftarrow$  un campione causale da  $\mathbf{P}(X_i|\text{genitori}(X_i))$ 
    return x

```

Figura 13.16 Un algoritmo di campionamento che genera eventi partendo da una rete bayesiana. Ogni variabile è campionata secondo la distribuzione condizionata dati i valori già campionati per i suoi genitori.

3. Campione da $\mathbf{P}(Pioggia|Coperto = true) = \langle 0,8, 0,2 \rangle$; il valore è *true*.
4. Campione da $\mathbf{P}(ErbaBagnata|Irrigatore = false, Pioggia = true) = \langle 0,9, 0,1 \rangle$; il valore è *true*.

In questo caso CAMPIONA-PRIORI restituisce l'evento $[true, false, true, true]$.

È facile constatare che CAMPIONA-PRIORI genera campioni dalla distribuzione congiunta a priori specificata dalla rete. Sia $S_{CP}(x_1, \dots, x_n)$ la probabilità che uno specifico evento sia generato dall'algoritmo CAMPIONA-PRIORI. *Semplicemente esaminando il processo di campionamento*, vediamo che

$$S_{CP}(x_1 \dots x_n) = \prod_{i=1}^n P(x_i | genitori(X_i))$$

dato che ogni passo di campionamento dipende solo dai valori dei genitori. Questa espressione dovrebbe risultare familiare, perché corrisponde anche alla probabilità dell'evento secondo la rappresentazione della distribuzione congiunta della rete bayesiana (Equazione (13.2)). Questo significa che

$$S_{CP}(x_1, \dots, x_n) = P(x_1, \dots, x_n).$$

Questa semplice equivalenza fa sì che risulti facile rispondere alle domande per mezzo del campionamento.

In ogni algoritmo di campionamento, le risposte sono calcolate contando i campioni effettivamente generati. Supponiamo che ci siano N campioni in totale prodotti dall'algoritmo CAMPIONA-PRIORI, e sia $N_{CP}(x_1, \dots, x_n)$ il numero di volte che lo specifico evento x_1, \dots, x_n si verifica nell'insieme dei campioni. Ci aspettiamo che il limite di questo numero, una frazione del totale, converga al suo valore atteso secondo la probabilità di campionamento:

$$\lim_{N \rightarrow \infty} \frac{N_{CP}(x_1, \dots, x_n)}{N} = S_{CP}(x_1, \dots, x_n) = P(x_1, \dots, x_n). \quad (13.6)$$

Consideriamo per esempio l'evento prodotto in precedenza: $[true, false, true, true]$. La probabilità di campionamento per quest'evento è

$$S_{CP}(true, false, true, true) = 0,5 \times 0,9 \times 0,8 \times 0,9 = 0,324.$$

Quindi, al tendere a infinito di N , ci aspettiamo che il 32,4% dei campioni appartenga a quest'evento.

D'ora in poi, ogni volta che useremo l'operatore di uguaglianza approssimata (“ \approx ”), lo faremo precisamente con questo significato: che la probabilità stimata diventa esatta al tendere all'infinito del numero di campioni. Una stima siffatta viene chiamata **consistente**. Per esempio, si può produrre una stima consistente della probabilità di qualsiasi evento parzialmente specificato x_1, \dots, x_m , con $m \leq n$, come segue:

$$P(x_1, \dots, x_m) \approx N_{CP}(x_1, \dots, x_m)/N. \quad (13.7)$$

Ovvero, la probabilità dell'evento può essere stimata come la frazione di tutti gli eventi completi generati dal processo di campionamento che corrispondono all'evento parzialmente specificato. Utilizzeremo \hat{P} (si pronuncia “P cappello”) per indicare una probabilità stimata. Così, se generiamo 1000 campioni della rete dell'irrigatore, e in 511 di essi risulta *Pioggia = true*, la probabilità stimata di *Pioggia* è $\hat{P}(Pioggia = true) = 0,511$.

Campionamento di rigetto nelle reti bayesiane

Il **campionamento di rigetto** (*rejection sampling*) è un metodo generale per ottenere informazioni da una distribuzione difficile da campionare data una distribuzione per cui ciò, al contrario, è facile. Nella sua forma più semplice può essere usato per calcolare probabilità condizionate, cioè per determinare $P(X|\mathbf{e})$. L'algoritmo CAMPIONAMENTO-RIGETTO, riportato nella Figura 13.17, per prima cosa genera campioni dalla distribuzione a priori specifici-

consistente

**campionamento
di rigetto**

```

function CAMPIONAMENTO-RIGETTO( $X, \mathbf{e}, bn, N$ ) returns una stima di  $P(X|\mathbf{e})$ 
  inputs:  $X$ , la variabile di query
     $\mathbf{e}$ , valori osservati per le variabili  $\mathbf{E}$ 
     $bn$ , una rete bayesiana
     $N$ , il numero totale di campioni da generare
  local variables:  $\mathbf{C}$ , un vettore di contatori per ogni valore di  $X$ , inizialmente a zero

  for  $j = 1$  to  $N$  do
     $\mathbf{x} \leftarrow \text{CAMPIONA-PRIORI}(bn)$ 
    if  $\mathbf{x}$  è consistente con  $\mathbf{e}$  then
       $\mathbf{C}[j] \leftarrow \mathbf{C}[j]+1$  dove  $x_j$  è il valore di  $X$  in  $\mathbf{x}$ 
  return NORMALIZZA( $\mathbf{C}$ )

```

Figura 13.17 L'algoritmo di campionamento di rigetto per calcolare la risposta a interrogazioni su reti bayesiane, data una evidenza.

cata dalla rete; quindi rifiuta tutti quelli che non corrispondono all'evidenza. Infine, ottiene la stima $\hat{P}(X = x | \mathbf{e})$ contando quante volte si verifica $X = x$ nei campioni rimanenti.

Sia $\hat{\mathbf{P}}(X | \mathbf{e})$ la distribuzione stimata restituita dall'algoritmo; questa distribuzione viene calcolata normalizzando $\mathbf{N}_{CP}(X, \mathbf{e})$, il vettore di conteggi campione per ogni valore di X dove il campione concorda con l'evidenza \mathbf{e} :

$$\hat{\mathbf{P}}(X | \mathbf{e}) = \alpha \mathbf{N}_{CP}(X, \mathbf{e}) = \frac{\mathbf{N}_{CP}(X, \mathbf{e})}{N_{CP}(\mathbf{e})}.$$

Dall'Equazione (13.7) questa diventa

$$\hat{\mathbf{P}}(X | \mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{\mathbf{P}(\mathbf{e})} = \mathbf{P}(X | \mathbf{e}).$$

Il campionamento di rigetto produce quindi una stima consistente della vera probabilità.

Continuando il nostro esempio della Figura 13.15(a), supponiamo di voler stimare $\mathbf{P}(\text{Pioggia} | \text{Irrigatore} = \text{true})$ con 100 campioni. Dei 100 generati, supponiamo che 73 siano rigettati perché $\text{Irrigatore} = \text{false}$, mentre in 27 risulta $\text{Irrigatore} = \text{true}$; dei 27, in 8 è $\text{Pioggia} = \text{true}$ e in 19 $\text{Pioggia} = \text{false}$. In definitiva,

$$\mathbf{P}(\text{Pioggia} | \text{Irrigatore} = \text{true}) \approx \text{NORMALIZZA}((8, 19)) = \langle 0,296, 0,704 \rangle.$$

La vera risposta è $\langle 0,3, 0,7 \rangle$: man mano che si raccolgono altri campioni, la stima convergerà su questi valori. La deviazione standard dell'errore in ogni probabilità sarà proporzionale a $1/\sqrt{n}$, dove n è il numero di campioni usati per la stima.

Ora che sappiamo che il campionamento di rigetto converge alla risposta corretta, ci chiediamo con quale velocità ciò accada. Più precisamente, quanti campioni sono richiesti prima di poter conoscere che le stime risultanti sono vicine alle risposte corrette con elevata probabilità? Mentre la complessità degli algoritmi esatti dipende in larga parte dalla topologia della rete – gli alberi sono facili da elaborare, le reti densamente connesse sono difficili – la complessità del campionamento di rigetto dipende principalmente dalla frazione di campioni accettati, e tale frazione è esattamente uguale alla probabilità a priori dell'evidenza, $P(\mathbf{e})$. Sfortunatamente, per problemi complessi con molte variabili di evidenza, questa frazione è molto piccola. In casi come la versione discreta della rete per le assicurazioni auto della Figura 13.9, la frazione di campioni consistenti con un tipico caso di evidenza campionati dalla

rete stessa è solitamente compresa tra uno su mille e uno su diecimila. La convergenza è estremamente lenta (Figura 13.19).

Ci aspettiamo che la frazione di campioni consistenti con l'evidenza \mathbf{e} diminuisca esponenzialmente al crescere del numero di variabili di evidenza, perciò la procedura è inapplicabile a problemi complessi e presenta difficoltà anche con variabili di evidenza a valori continui, perché la probabilità di produrre un campione consistente con tale evidenza è zero (se la variabile è realmente a valori continui) o infinitesimale (se è semplicemente un numero in virgola mobile a precisione finita).

Notate che il campionamento di rigetto è molto simile alla stima delle probabilità condizionate nel mondo reale. Per esempio, per stimare la probabilità condizionata che un essere umano sopravviva dopo l'impatto sul nostro pianeta di un asteroide di 1 km di diametro, si potrebbe semplicemente contare quante volte un essere umano sopravvive dopo un impatto di quel tipo, ignorando tutti i giorni in cui tale evento non si verifica (qui l'universo stesso assume il ruolo dell'algoritmo di generazione dei campioni). Per ottenere una stima decente, potrebbe essere necessario attendere 100 eventi di quel tipo, e naturalmente questo richiederebbe un tempo lunghissimo: questo è proprio il punto debole del campionamento di rigetto.

Campionamento di importanza

**campionamento
di importanza**

La tecnica statistica generale del **campionamento di importanza** cerca di emulare l'effetto del campionamento da una distribuzione P usando campioni da un'altra distribuzione Q . Ci assicuriamo che le risposte siano corrette al limite applicando un fattore di correzione $P(\mathbf{x})/Q(\mathbf{x})$, noto anche come **peso**, a ogni campione \mathbf{x} nel conteggio dei campioni.

Il motivo per cui si ricorre al campionamento di importanza nelle reti bayesiane è semplice: in realtà vorremmo campionare dalla distribuzione a posteriori reale condizionata su tutte le evidenze, ma questo solitamente è troppo difficile;⁶ perciò ci limitiamo a campionare da una distribuzione facile e applichiamo le necessarie correzioni. Anche il motivo per cui il campionamento di importanza funziona è semplice: indichiamo con \mathbf{Z} le variabili non di evidenza; se potessimo campionare direttamente da $P(\mathbf{z} | \mathbf{e})$, potremmo costruire stime come la seguente:

$$\hat{P}(\mathbf{z} | \mathbf{e}) = \frac{N_P(\mathbf{z})}{N} \approx \hat{P}(\mathbf{z} | \mathbf{e})$$

dove $N_P(\mathbf{z})$ è il numero di campioni con $\mathbf{Z} = \mathbf{z}$ nel campionamento da P . Ora supponiamo invece di campionare da $Q(\mathbf{s})$. In questo caso la stima include i fattori di correzione:

$$\hat{P}(\mathbf{z} | \mathbf{e}) = \frac{N_Q(\mathbf{z})}{N} \frac{P(\mathbf{z} | \mathbf{e})}{Q(\mathbf{z})} \approx Q(\mathbf{z}) \frac{P(\mathbf{z} | \mathbf{e})}{Q(\mathbf{z})} = P(\mathbf{z} | \mathbf{e}).$$

Quindi, la stima converge al valore corretto *indipendentemente da quale distribuzione di campionamento Q si utilizzi* (il solo requisito tecnico è che $Q(\mathbf{z})$ non sia zero per alcun \mathbf{z} dove $P(\mathbf{z} | \mathbf{e})$ è diversa da zero). Intuitivamente, il fattore di correzione compensa per il sovraccampionamento. Per esempio, se $Q(\mathbf{z})$ è molto più grande di $P(\mathbf{z} | \mathbf{e})$ per qualche \mathbf{z} , i campioni di \mathbf{z} saranno molti più del dovuto, ma ognuno avrà un peso piccolo, perciò è come se fossero nel numero giusto.

Per quanto riguarda la Q da usare, ne vogliamo una che sia facile da campionare e più vicina possibile alla probabilità a posteriori reale $P(\mathbf{z} | \mathbf{e})$. L'approccio più comune è chiamato **pesatura di verosimiglianza** (per motivi che vedremo tra breve). Come si vede nella funzione

**pesatura
di verosimiglianza**

⁶ Se fosse facile, potremmo approssimare la probabilità desiderata con accuratezza arbitraria mediante un numero polinomiale di campioni. Si può dimostrare che non può esistere un tale schema di approssimazione in tempo polinomiale.

```

function PESATURA-VEROSIMIGLIANZA ( $X, \mathbf{e}, bn, N$ ) returns una stima di  $\mathbf{P}(X|\mathbf{e})$ 
  inputs:  $X$ , la variabile di query
     $\mathbf{e}$ , i valori osservati per le variabili  $\mathbf{E}$ 
     $bn$ , una rete bayesiana che specifica la distribuzione congiunta  $\mathbf{P}(X_1, \dots, X_n)$ 
     $N$ , il numero totale di campioni da generare
  local variables:  $\mathbf{W}$ , un vettore di contatori pesati per ogni valore di  $X$ , inizialmente a zero

  for  $j = 1$  to  $N$  do
     $\mathbf{x}, w \leftarrow \text{CAMPIONE-PESATO}(bn, \mathbf{e})$ 
     $\mathbf{W}[j] \leftarrow \mathbf{W}[j] + w$  dove  $x_j$  è il valore di  $X$  in  $\mathbf{x}$ 
  return NORMALIZZA( $\mathbf{W}$ )

function CAMPIONE-PESATO( $bn, \mathbf{e}$ ) returns un evento e un peso
   $w \leftarrow 1; \mathbf{x} \leftarrow$  un evento con  $n$  elementi con valori fissati da  $\mathbf{e}$ 
  for  $i = 1$  to  $n$  do
    if  $X_i$  è una variabile di evidenza con valore  $x_{ij}$  in  $\mathbf{e}$ 
      then  $w \leftarrow w \times P(X_i = x_{ij} | \text{genitori}(X_i))$ 
      else  $x[i] \leftarrow$  un campione casuale da  $\mathbf{P}(X_i | \text{genitori}(X_i))$ 
  return  $\mathbf{x}, w$ 

```

Figura 13.18 L'algoritmo di pesatura di verosimiglianza per l'inferenza nelle reti bayesiane. In CAMPIONE-PESATO, ogni variabile non di evidenza è campionata secondo la distribuzione condizionata dati i valori già campionati per i suoi genitori, mentre si accumula un peso basato sulla probabilità per ogni variabile di evidenza.

CAMPIONE-PESATO della Figura 13.18, l'algoritmo fissa i valori per le variabili di evidenza \mathbf{E} e campiona tutte le variabili non di evidenza in ordine topologico, ciascuna condizionata ai suoi genitori. Così si garantisce che ogni evento generato sia consistente con l'evidenza.

Chiamiamo Q_{PV} la distribuzione campionaria prodotta da questo algoritmo. Se le variabili non di evidenza sono $\mathbf{Z} = \{Z_1, \dots, Z_l\}$, allora abbiamo:

$$Q_{PV}(\mathbf{z}) = \prod_{i=1}^l P(z_i | \text{genitori}(Z_i)) \quad (13.8)$$

poiché ogni variabile è campionata condizionatamente ai suoi genitori. Per completare l'algoritmo dobbiamo sapere come calcolare il peso per ciascun campione generato da Q_{PV} . Secondo lo schema generale del campionamento di importanza, il peso dev'essere:

$$w(\mathbf{z}) = P(\mathbf{z} | \mathbf{e}) / Q_{PV}(\mathbf{z}) = \alpha P(\mathbf{z}, \mathbf{e}) / Q_{PV}(\mathbf{z})$$

dove il fattore di normalizzazione $\alpha = 1/P(\mathbf{e})$ è lo stesso per tutti i campioni. Ora \mathbf{z} ed \mathbf{e} insieme coprono tutte le variabili della rete bayesiana, perciò $P(\mathbf{z}, \mathbf{e})$ è semplicemente il prodotto di tutte le probabilità condizionate (Equazione (13.2)) e possiamo scriverlo come prodotto delle probabilità condizionate per le variabili non di evidenza per il prodotto delle probabilità condizionate per le variabili di evidenza:

$$\begin{aligned} w(\mathbf{z}) &= \alpha \frac{P(\mathbf{z}, \mathbf{e})}{Q_{PV}(\mathbf{z})} = \alpha \frac{\prod_{i=1}^l P(z_i | \text{genitori}(Z_i)) \prod_{i=1}^m P(e_i | \text{genitori}(E_i))}{\prod_{i=1}^l P(z_i | \text{genitori}(Z_i))} \\ &= \alpha \prod_{i=1}^m P(e_i | \text{genitori}(E_i)). \end{aligned} \quad (13.9)$$

Il peso, quindi, è il prodotto delle probabilità condizionate per le variabili di evidenza dati i loro genitori (la probabilità di variabili dell'evidenza è generalmente chiamata **verosimiglianza**, da cui il termine pesatura di verosimiglianza). Il calcolo del peso è implementato in modo incrementale nella funzione CAMPIONE-PESATO, moltiplicando per la probabilità condizionata ogni volta che si incontra una variabile di evidenza. La normalizzazione è effettuata alla fine prima di restituire il risultato dell'interrogazione.

Applichiamo ora l'algoritmo alla rete della Figura 13.15(a), utilizzando la query $\mathbf{P}(\text{Pioggia}|\text{Coperto} = \text{true}, \text{ErbaBagnata} = \text{true})$ e l'ordinamento *Coperto, Irrigatore, Pioggia, ErbaBagnata* (andrà bene qualsiasi ordinamento topologico). Il processo si svolge come segue: per prima cosa si pone il valore del peso w a 1,0; quindi viene generato un evento.

1. *Coperto* è una variabile di evidenza con valore *true*, quindi assegniamo:

$$w \leftarrow w \times P(\text{Coperto} = \text{true}) = 0,5 .$$

2. *Irrigatore* non è una variabile di evidenza, perciò campioniamo da $\mathbf{P}(\text{Irrigatore}|\text{Coperto} = \text{true}) = \langle 0,1, 0,9 \rangle$; supponiamo di ottenere *false*.
3. *Pioggia* non è una variabile di evidenza, perciò campioniamo da $\mathbf{P}(\text{Pioggia}|\text{Coperto} = \text{true}) = \langle 0,8, 0,2 \rangle$; supponiamo di ottenere *true*.
4. *ErbaBagnata* è una variabile di evidenza con valore *true*. Quindi assegniamo

$$w \leftarrow w \times P(\text{ErbaBagnata} = \text{true}|\text{Irrigatore} = \text{false}, \text{Pioggia} = \text{true}) = 0,5 \times 0,9 = 0,45 .$$

Qui CAMPIONE-PESATO restituisce l'evento[*true, false, true, true*] con peso 0,45, che viene conteggiato sotto *Pioggia = true*.

Notate che $Genitori(Z_i)$ può includere sia variabili non di evidenza che variabili di evidenza. A differenza della distribuzione a priori $P(\mathbf{z})$, la distribuzione Q_{PV} considera in qualche modo le evidenze: i valori campionati per ogni Z_i sono influenzati dalle evidenze incluse tra gli antenati di Z_i . Per esempio, nel campionamento di *Irrigatore* l'algoritmo presta attenzione alla evidenza *Coperto = true* nella sua variabile genitore. D'altra parte, Q_{PV} non è influenzata dalle evidenze quanto la vera distribuzione a posteriori $P(\mathbf{z}|\mathbf{e})$, perché i valori campionati per ogni Z_i ignorano le evidenze che non sono presenti tra gli antenati di Z_i . Per esempio, nel campionamento di *Irrigatore* e *Pioggia* l'algoritmo ignora l'evidenza nella variabile figlia *ErbaBagnata = true*; ciò significa che genererà molti campioni con *Irrigatore = false* e *Pioggia = false* nonostante il fatto che le evidenze escludano in effetti questo caso. Tali campioni avranno peso zero.

Dato che utilizza tutti i campioni generati, la pesatura di verosimiglianza può risultare molto più efficiente del campionamento di rigetto. Tuttavia, le prestazioni caleranno al crescere del numero delle variabili di evidenza. Ciò è dovuto al fatto che la maggior parte dei campioni avrà un peso molto basso, quindi la stima pesata sarà dominata dalla piccola frazione di campioni che accordano alle evidenze una verosimiglianza maggiore di un infinitesimo. Il problema è exacerbato quando le variabili di evidenza sono presenti nella parte finale dell'ordinamento, perché in questi casi le variabili non di evidenza non avranno evidenze nei loro genitori che possano guidare la generazione di campioni. Ciò significa che i campioni saranno mere allucinazioni, ovvero simulazioni con poca somiglianza alla realtà suggerita dalle evidenze.

Nell'applicazione alla versione discreta della rete per le assicurazioni auto illustrata nella Figura 13.9, la pesatura di verosimiglianza è nettamente più efficiente del campionamento di rigetto (Figura 13.19). La rete per le assicurazioni auto è un caso relativamente favorevole per la pesatura di verosimiglianza perché gran parte delle evidenze è situata nella parte iniziale dell'ordinamento e le variabili di query sono nodi foglia.

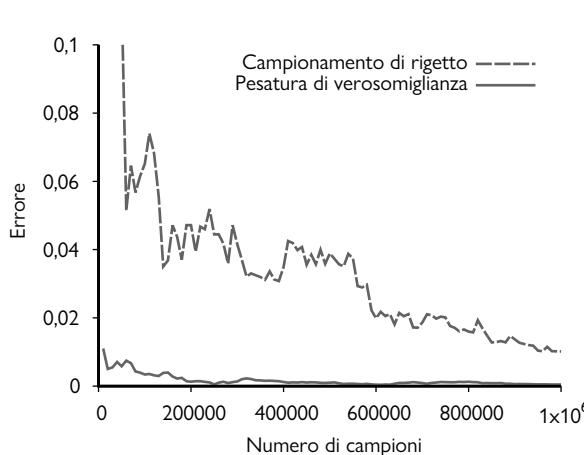


Figura 13.19 Prestazioni degli algoritmi di campionamento di rigetto e di pesatura di verosomiglianza sulla rete delle assicurazioni auto. L'asse delle ascisse riporta il numero di campioni generati e quello delle ordinate riporta il massimo errore assoluto in ognuno dei valori di probabilità per una query su *CostoProprietà*.

13.4.2 Inferenza mediante simulazione con catene di Markov

Gli algoritmi **Monte Carlo per catene di Markov** (MCMC) operano in modo diverso dal campionamento di rigetto e dalla pesatura di verosomiglianza. Anziché generare ogni campione da zero, infatti, MCMC genera un campione apportando una modifica casuale a quello precedente. È quindi utile pensare che l'algoritmo MCMC sia in un particolare *stato corrente* che specifica un valore per ogni variabile e generi uno *stato successivo* apportando modifiche casuali allo stato corrente.

Il termine **catena di Markov** fa riferimento a un processo casuale che genera una sequenza di stati (le catene di Markov sono trattate anche nei Capitoli 14 e 17; anche l'algoritmo di simulated annealing trattato nel Capitolo 4 e l'algoritmo WALKSAT nel Capitolo 7 fanno parte della famiglia degli algoritmi MCMC). Iniziamo descrivendo una particolare forma di MCMC chiamata **campionamento di Gibbs**, particolarmente adatta alle reti bayesiane, poi passeremo al più generale algoritmo **Metropolis–Hastings**, che consente una flessibilità molto maggiore nella generazione di campioni.

Campionamento di Gibbs in reti bayesiane

L'algoritmo di campionamento di Gibbs per reti bayesiane inizia con uno stato arbitrario (con le variabili di evidenza fissate ai loro valori osservati) e genera uno stato successivo campionando casualmente un valore per una delle variabili non di evidenza X_i . Ricordiamo dal Paragrafo 13.2.1 che X_i è indipendente da tutte le altre variabili data la sua coperta di Markov (i suoi genitori, i suoi figli e gli altri genitori dei figli); quindi il campionamento di Gibbs per X_i è un campionamento *condizionato ai valori correnti delle variabili nella sua coperta di Markov*. L'algoritmo si muove casualmente nello spazio degli stati – lo spazio dei possibili assegnamenti completi – modificando una variabile per volta ma tenendo fissate le variabili di evidenza. L'algoritmo completo è mostrato nella Figura 13.20.

Considerate la query $\mathbf{P}(\text{Pioggia}|\text{Irrigatore} = \text{true}, \text{ErbaBagnata} = \text{true})$ per la rete della Figura 13.15(a). Le variabili di evidenza *Irrigatore* ed *ErbaBagnata* rimangono fissate ai valori osservati (entrambi *true*), mentre quelle non di evidenza *Coperto* e *Pioggia* sono inizializzate casualmente: diciamo ai valori *true* e *false*, rispettivamente. Così, lo stato iniziale è **[true, true, false, true]**, dove abbiamo evidenziato in grassetto i valori di evidenza fissati. A questo

Monte Carlo per catene di Markov

catena di Markov

campionamento di Gibbs
Metropolis–Hastings

```

function GIBBS-ASK( $X, \mathbf{e}, bn, N$ ) returns una stima di  $\mathbf{P}(X|\mathbf{e})$ 
local variables:  $C$ , un vettore di contatori per ogni valore di  $X$ , inizialmente a zero
 $Z$ , le variabili non di evidenza in  $bn$ 
 $x$ , lo stato corrente della rete, inizializzato con i valori di  $\mathbf{e}$ 

inizializza  $x$  con valori casuali delle variabili in  $Z$ 
for  $k = 1$  to  $N$  do
    sceglie una variabile  $Z_i$  da  $Z$  secondo una qualsiasi distribuzione  $\rho(i)$ 
    assegna il valore di  $Z_i$  in  $x$  campionandolo da  $\mathbf{P}(Z_i|mb(Z_i))$ 
     $C[j] \leftarrow C[j] + 1$  dove  $x_j$  è il valore di  $X$  in  $x$ 
return NORMALIZZA( $C$ )

```

Figura 13.20 L'algoritmo di campionamento di Gibbs per l'inferenza approssimata nelle reti bayesiane. Questa versione sceglie le variabili a caso, ma l'algoritmo funziona anche nella versione con un ciclo sulle variabili.

punto si esegue ripetutamente il campionamento delle variabili non di evidenza Z_i in ordine casuale secondo una distribuzione di probabilità $\rho(i)$ per scegliere le variabili.

Per esempio:

1. *Coperto* viene scelta e poi campionata, dati i valori correnti delle variabili nella sua coperta di Markov: in questo caso campioniamo da $\mathbf{P}(Coperto|Irrigatore = true, Pioggia = false)$. Supponiamo che il risultato sia *Coperto = false*. Il nuovo stato corrente sarà [*false, true, false, true*].
2. *Pioggia* viene scelta e poi campionata, dati i valori correnti delle variabili nella sua coperta di Markov: in questo caso campioniamo da $\mathbf{P}(Pioggia|Coperto = false, Irrigatore = true, ErbaBagnata = true)$. Supponiamo che il risultato sia *Pioggia = true*. Il nuovo stato corrente sarà [*false, true, true, true*].

L'unico dettaglio che rimane riguarda il metodo per calcolare la distribuzione della coperta di Markov $\mathbf{P}(X_i | mb(X_i))$, dove $mb(X_i)$ denota i valori delle variabili nella coperta di Markov di X_i , $MB(X_i)$. Fortunatamente non serve inferenza complessa. Come è mostrato nell'Esercizio 13.MARB, la distribuzione è data da:

$$P(x_i | mb(X_i)) = \alpha P(x_i | genitori(X_i)) \prod_{Y_j \in Figli(X_i)} P(y_j | genitori(Y_j)). \quad (13.10)$$

In altre parole, per ogni valore x_i , la probabilità si ottiene moltiplicando le probabilità dalle CPT di X_i e dei suoi figli. Per esempio, nel primo passo di campionamento mostrato sopra abbiamo campionato da $\mathbf{P}(Coperto | Irrigatore = true, Pioggia = false)$. Per l'Equazione (13.10), e abbreviando i nomi delle variabili, abbiamo:

$$\begin{aligned} P(c | i, \neg p) &= \alpha P(c)P(i | c)P(\neg p | c) = \alpha 0,5 \cdot 0,1 \cdot 0,2 \\ P(\neg c | i, \neg p) &= \alpha P(\neg c)P(i | \neg c)P(\neg p | \neg c) = \alpha 0,5 \cdot 0,5 \cdot 0,8, \end{aligned}$$

quindi la distribuzione di campionamento è $\alpha \langle 0,001, 0,020 \rangle \approx \langle 0,048, 0,952 \rangle$.

La Figura 13.21(a) mostra la catena di Markov completa per il caso in cui le variabili sono scelte in modo uniforme, cioè $\rho(Coperto) = \rho(Pioggia) = 0,5$. L'algoritmo non fa altro che muoversi attraverso il grafo, seguendo gli archi con le probabilità stabilite. Ogni stato visitato durante questo processo è un campione che contribuisce alla stima per la variabile query *Pioggia*. Se il processo visita 20 stati in cui *Pioggia* è vera e 60 stati in cui *Pioggia* è falsa, allora la risposta alla query è $\text{NORMALIZZA}(\langle 20, 60 \rangle) = \langle 0,25, 0,75 \rangle$.

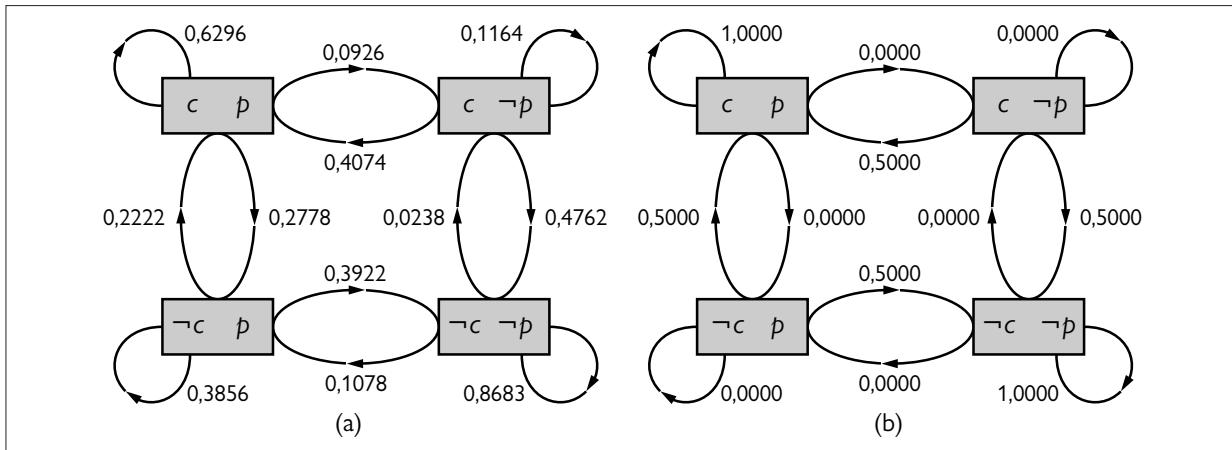


Figura 13.21 (a) Stati e probabilità di transizione della catena di Markov per la query $\mathbf{P}(\text{Pioggia} \mid \text{Irrigatore} = \text{true}, \text{ErbaBagnata} = \text{true})$. Notate i cicli ad anello su uno stesso nodo: lo stato rimane invariato quando viene scelta qualunque delle due variabili e poi ricampiona lo stesso valore che ha già. (b) Probabilità di transizione quando per la CPT *Pioggia* deve avere lo stesso valore di *Coperto*.

Analisi delle catene di Markov

Abbiamo detto che il campionamento di Gibbs opera percorrendo casualmente lo spazio degli stati per generare campioni. Per spiegare il motivo per cui il campionamento di Gibbs funziona *correttamente*, ovvero perché le sue stime convergono al limite ai valori corretti, abbiamo bisogno di un’analisi attenta (questo paragrafo presenta una trattazione matematica e può eventualmente essere saltato in una prima lettura).

Iniziamo con alcuni dei concetti di base per analizzare le catene di Markov in generale. Ogni catena di Markov è definita dal suo stato iniziale e dal suo **nucleo di transizione** $k(\mathbf{x} \rightarrow \mathbf{x}')$ – la probabilità di una transizione nello stato \mathbf{x}' partendo dallo stato \mathbf{x} . Ora supponiamo di eseguire la catena di Markov per t passi, e sia $\pi_t(\mathbf{x})$ la probabilità che il sistema sia nello stato \mathbf{x} al tempo t . Similmente, sia $\pi_{t+1}(\mathbf{x}')$ la probabilità di essere nello stato \mathbf{x}' al tempo $t+1$. Data $\pi_t(\mathbf{x})$, possiamo calcolare $\pi_{t+1}(\mathbf{x}')$ sommando su tutti gli stati \mathbf{x} in cui il sistema potrebbe trovarsi al tempo t la probabilità di essere in \mathbf{x} per la probabilità di fare la transizione in \mathbf{x}' :

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) k(\mathbf{x} \rightarrow \mathbf{x}').$$

Diciamo che la catena ha raggiunto la sua **distribuzione stazionaria** se $\pi_t = \pi_{t+1}$. Chiamiamo π tale distribuzione stazionaria; l’equazione che la definisce è:

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) k(\mathbf{x} \rightarrow \mathbf{x}') \quad \text{per tutti gli stati } \mathbf{x}'. \quad (13.11)$$

Purché il nucleo di transizione k sia **ergodico** – il che significa che ogni stato è raggiungibile da ogni altro e non ci sono cicli strettamente periodici – esiste esattamente una distribuzione π che soddisfa questa equazione per ogni k dato.

L’Equazione (13.11) afferma che il “flusso in uscita” atteso da ogni stato (cioè la sua “popolazione” corrente) è uguale al “flusso in entrata” atteso da tutti gli stati. Un modo ovvio per soddisfare questa relazione si ha se il flusso atteso tra ogni coppia di stati è identico in entrambe le direzioni, cioè:

$$\pi(\mathbf{x}) k(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') k(\mathbf{x}' \rightarrow \mathbf{x}) \quad \text{per tutti gli } \mathbf{x}, \mathbf{x}'. \quad (13.12)$$

Quando valgono queste equazioni, diciamo che $k(\mathbf{x} \rightarrow \mathbf{x}')$ è in **equilibrio dettagliato** con $\pi(\mathbf{x})$. Un caso speciale è quello di un auto-anello $\mathbf{x} = \mathbf{x}'$, cioè una transizione da uno stato a se stesso.

**nucleo
di transizione**

**distribuzione
stazionaria**

ergodico

**equilibrio
dettagliato**

In questo caso, la condizione di equilibrio dettagliato diventa $\pi(\mathbf{x})k(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}')k(\mathbf{x}' \rightarrow \mathbf{x})$ che naturalmente è vera per qualsiasi distribuzione stazionaria π e qualsiasi nucleo di transizione k .

Possiamo dimostrare che l'equilibrio dettagliato implica la stazionarietà semplicemente sommando su \mathbf{x} nell'Equazione (13.12). Abbiamo:

$$\sum_{\mathbf{x}} \pi(\mathbf{x})k(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}')k(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{k}} k(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

dove l'ultimo passaggio segue perché si ha la garanzia che si verifichi una transizione da \mathbf{x}' .

Perché il campionamento di Gibbs funziona



Mostriremo ora che il campionamento di Gibbs fornisce stime consistenti per le probabilità a posteriori. La tesi di base è semplice: *la distribuzione stazionaria del processo di campionamento di Gibbs è esattamente la distribuzione a posteriori per le variabili non di evidenza condizionate all'evidenza*. Questa importante proprietà segue dal modo specifico in cui il processo di campionamento di Gibbs passa da uno stato all'altro.

La definizione generale di campionamento di Gibbs è la seguente: una variabile X_i viene scelta e poi campionata condizionatamente sui valori correnti di *tutte* le altre variabili (nel caso specifico delle reti bayesiane, sfruttiamo il fatto aggiuntivo che campionare condizionatamente su tutte le variabili equivale a campionare condizionatamente sulla coperta di Markov della variabile, come si è visto nel Paragrafo 13.2.1). Utilizzeremo la notazione $\bar{\mathbf{x}}_i$ per fare riferimento a queste altre variabili (eccetto le variabili di evidenza); i loro valori nello stato corrente sono $\bar{\mathbf{x}}_i$.

Per scrivere il nucleo di transizione $k(\mathbf{x} \rightarrow \mathbf{x}')$ per il campionamento di Gibbs occorre considerare tre casi:

1. Gli stati \mathbf{x} e \mathbf{x}' differiscono in due o più variabili. In questo caso, $k(\mathbf{x} \rightarrow \mathbf{x}') = 0$ perché il campionamento di Gibbs cambia solo una singola variabile.
2. Gli stati differiscono in esattamente una variabile X_i che cambia il suo valore da x_i a x'_i . La probabilità di un tale evento è:

$$k(\mathbf{x} \rightarrow \mathbf{x}') = k((x_i, \bar{\mathbf{x}}_i) \rightarrow (x'_i, \bar{\mathbf{x}}_i)) = \rho(i) P(x'_i | \bar{\mathbf{x}}_i). \quad (13.13)$$

3. Gli stati coincidono: $\mathbf{x} = \mathbf{x}'$. In questo caso, *qualsiasi* variabile potrebbe essere scelta, ma poi il processo di campionamento produrrebbe lo stesso valore che la variabile possiede già. La probabilità di un tale evento è:

$$k(\mathbf{x} \rightarrow \mathbf{x}) = \sum_i \rho(i) k((x_i, \bar{\mathbf{x}}_i) \rightarrow (x_i, \bar{\mathbf{x}}_i)) = \sum_i \rho(i) P(x_i | \bar{\mathbf{x}}_i).$$

Ora mostriamo che questa definizione generale di campionamento di Gibbs soddisfa l'equazione dell'equilibrio dettagliato con una distribuzione stazionaria uguale a $P(\mathbf{x} | \mathbf{e})$, la vera distribuzione a posteriori sulle variabili non di evidenza. Mostriamo quindi che $\pi(\mathbf{x})k(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}')k(\mathbf{x}' \rightarrow \mathbf{x})$ dove $\pi(\mathbf{x}) = P(\mathbf{x} | \mathbf{e})$, per tutti gli stati \mathbf{x} e \mathbf{x}' .

Per il primo e il terzo caso descritto sopra, l'equilibrio dettagliato è *sempre* soddisfatto: se due stati differiscono in due o più variabili, la probabilità di transizione in entrambe le direzioni è zero. Se $\mathbf{x} \neq \mathbf{x}'$, allora dall'Equazione (13.13) abbiamo:

$$\begin{aligned} \pi(\mathbf{x})k(\mathbf{x} \rightarrow \mathbf{x}') &= P(\mathbf{x} | \mathbf{e})\rho(i)P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) = \rho(i)P(x_i, \bar{\mathbf{x}}_i | \mathbf{e})P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \\ &= \rho(i)P(x_i | \bar{\mathbf{x}}_i, \mathbf{e})P(\bar{\mathbf{x}}_i | \mathbf{e})P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \quad (\text{usando la regola della catena sul primo termine}) \\ &= \rho(i)P(x_i | \bar{\mathbf{x}}_i, \mathbf{e})P(x'_i, \bar{\mathbf{x}}_i | \mathbf{e}) \quad (\text{inversa della regola della catena sugli ultimi due termini}) \\ &= \pi(\mathbf{x}')k(\mathbf{x}' \rightarrow \mathbf{x}). \end{aligned}$$

L'ultimo tassello del mosaico è l'ergodicità della catena: ogni stato deve essere raggiungibile da ogni altro e non devono esserci cicli periodici. Entrambe le condizioni sono soddisfatte purché le CPT non contengano probabilità pari a 0 o 1. La raggiungibilità deriva dal fatto che possiamo convertire uno stato in un altro cambiando una variabile alla volta, mentre l'assenza di cicli periodici deriva dal fatto che ogni stato ha un auto-anello che torna su se stesso con probabilità non nulla. Quindi, nelle condizioni citate, k è ergodico, il che significa che i campioni generati dal campionamento di Gibbs saranno tratti alla fine dalla vera distribuzione a posteriori.

Complessità del campionamento di Gibbs

Per prima cosa notiamo un aspetto positivo: ogni passo del campionamento di Gibbs comporta il calcolo della distribuzione della coperta di Markov per la variabile scelta X_i , che richiede un numero di moltiplicazioni proporzionale al numero di figli di X_i e alla dimensione dell'intervallo di X_i . Questo punto è importante perché significa che *il lavoro richiesto per generare ogni campione è indipendente dalla dimensione della rete*.



E ora passiamo alle cattive notizie: la complessità del campionamento di Gibbs è molto più difficile da analizzare di quella del campionamento di rigetto e della pesatura di verosimiglianza. Il primo aspetto è che il campionamento di Gibbs, a differenza della pesatura di verosimiglianza, *presta* attenzione alle evidenze che seguono nell'ordinamento topologico. Le informazioni si propagano dai nodi di evidenza in tutte le direzioni: prima ogni nodo vicino ai nodi di evidenza campiona valori che riflettono l'evidenza di quei nodi; poi si passa ai vicini dei vicini, e così via. Ci aspettiamo quindi che il campionamento di Gibbs offra prestazioni migliori della pesatura di verosimiglianza quando le evidenze sono per lo più alla fine dell'ordinamento topologico; e in effetti ne vediamo la conferma nella Figura 13.22.

La velocità di convergenza per il campionamento di Gibbs – la **velocità di mixing** della catena di Markov definita dall'algoritmo – dipende fortemente dalle proprietà quantitative delle distribuzioni condizionate nella rete. Per capirlo, considerate ciò che accade nella Figura 13.15(a) quando la CPT per *Pioggia* diventa deterministica: piove se e solo se è coperto. In tal caso, la vera distribuzione a posteriori per la query $\mathbf{P}(\text{Pioggia} \mid \text{irrigatore}, \text{erbabagnata})$ è approssimativamente $\langle 0,18, 0,82 \rangle$ ma il campionamento di Gibbs non raggiungerà mai tale valore. Il problema è che i due soli stati congiunti per *Coperto* e *Pioggia* che hanno probabilità non nulla sono *[true, true]* e *[false, false]*. Partendo da *[true, true]*, la catena non può

velocità di mixing

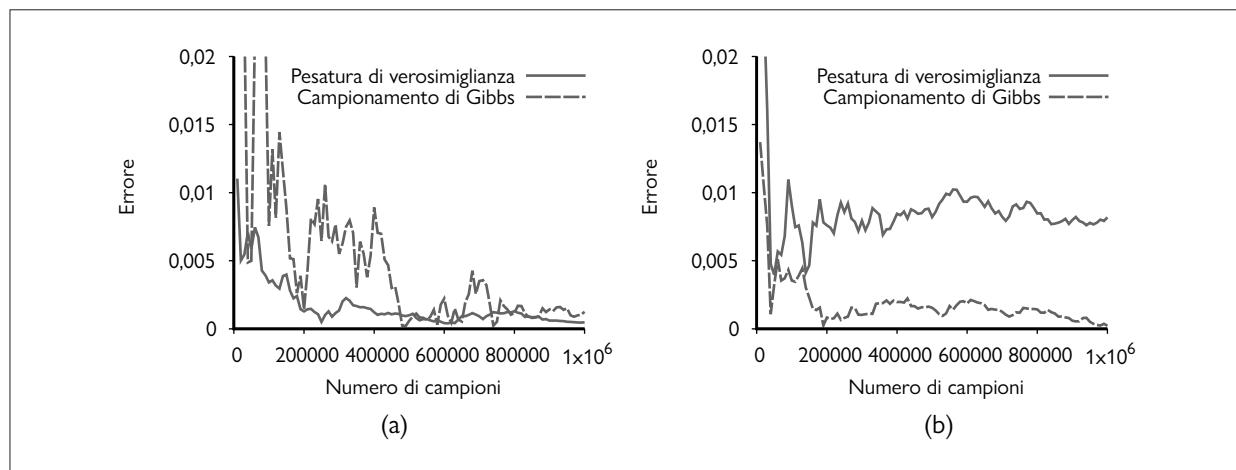


Figura 13.22 Prestazioni del campionamento di Gibbs rispetto alla pesatura di verosimiglianza sulla rete delle assicurazioni auto: (a) per la query standard su *CostoProprietà* e (b) per il caso in cui le variabili di output sono osservate ed *Età* è la variabile di query.

mai raggiungere $[false, false]$, perché le transizioni verso gli stati intermedi hanno probabilità zero (cfr. Figura 13.21(b)). Quindi, se il processo inizia in $[true, true]$ riporta sempre una probabilità a posteriori per la query di $\langle 1, 0, 0, 0 \rangle$; se inizia in $[false, false]$ riporta sempre una probabilità a posteriori per la query di $\langle 0, 0, 1, 0 \rangle$.

Il campionamento di Gibbs fallisce in questo caso perché la relazione deterministica tra *Coperto* e *Pioggia* viola la proprietà di ergodicità richiesta per la convergenza. Tuttavia, se rendiamo la relazione *quasi* deterministica, allora si riavrebbe la convergenza, che però sarebbe arbitrariamente lenta. Esistono diversi interventi che facilitano un mixing più veloce negli algoritmi MCMC: uno è il **campionamento a blocchi**, che significa campionare più variabili contemporaneamente. In questo caso potremmo campionare *Coperto* e *Pioggia* in modo congiunto, condizionatamente alla loro coperta di Markov combinata. Un altro metodo è quello di generare gli stati successivi in modo più intelligente, come vedremo nel prossimo paragrafo.

campionamento a blocchi

Il campionamento di Metropolis–Hastings

Il metodo di campionamento di Metropolis–Hastings o MH è forse quello di più ampia applicabilità tra gli algoritmi di tipo MCMC. Come il campionamento di Gibbs, il campionamento MH è progettato in modo da generare campioni \mathbf{x} (alla fine) secondo le probabilità target $\pi(\mathbf{x})$; nel caso dell’inferenza in reti bayesiane, vogliamo $\pi(\mathbf{x}) = P(\mathbf{x} \mid \mathbf{e})$. Come il simulated annealing (Paragrafo 4.1.2 del Capitolo 4), MH ha due fasi in ciascuna iterazione del processo di campionamento:

1. Campionare un nuovo stato \mathbf{x}' da una **distribuzione proposta** $q(\mathbf{x}' \mid \mathbf{x})$, dato lo stato corrente \mathbf{x} .
2. Accettare probabilisticamente o rifiutare \mathbf{x}' secondo la **probabilità di accettazione**

$$a(\mathbf{x}' \mid \mathbf{x}) = \min \left(1, \frac{\pi(\mathbf{x}') q(\mathbf{x} \mid \mathbf{x}')}{\pi(\mathbf{x}) q(\mathbf{x}' \mid \mathbf{x})} \right).$$

Se la proposta è respinta, allora lo stato rimane in \mathbf{x} .

Il nucleo di transizione per il campionamento MH è costituito da questo processo in due passaggi. Notate che, se la proposta è rifiutata, la catena rimane nello stesso stato.

La distribuzione proposta è responsabile, come indica il nome, per proporre uno stato successivo \mathbf{x}' . Per esempio, $q(\mathbf{x}' \mid \mathbf{x})$ potrebbe essere definito come segue:

- con probabilità 0,95, esegui un campionamento di Gibbs per generare \mathbf{x}' .
- Altrimenti, genera direttamente \mathbf{x}' eseguendo l’algoritmo CAMPIONE-PESATO già incontrato nella Figura 14.14.

Con questa distribuzione proposta il campionamento MH esegue circa 20 passi di campionamento di Gibbs, poi “riavvia” il processo da un nuovo stato (supponendo che sia accettato) che è stato generato da zero. Con questo stratagemma si aggira il problema del campionamento di Gibbs che si blocca in una parte dello spazio degli stati e non è in grado di raggiungere le altre parti.

Potreste chiedervi come mai sappiamo che il campionamento MH con una distribuzione proposta così particolare converga effettivamente alla soluzione corretta. La particolarità del campionamento MH è che *la convergenza alla distribuzione stazionaria corretta è garantita per ogni distribuzione proposta*, purché il nucleo di transizione risultante sia ergodico.

Questa proprietà segue dal modo in cui è definita la probabilità di accettazione. Come per il campionamento di Gibbs, l’auto-anello da uno stato allo stesso stato con $\mathbf{x} = \mathbf{x}'$ soddisfa automaticamente l’equilibrio dettagliato, perciò possiamo focalizzarci sul caso in cui $\mathbf{x} \neq \mathbf{x}'$.



distribuzione proposta

probabilità di accettazione

Questo può verificarsi soltanto se la proposta è accettata. La probabilità che si verifichi una tale transizione è:

$$k(\mathbf{x} \rightarrow \mathbf{x}') = q(\mathbf{x}' | \mathbf{x})a(\mathbf{x}' | \mathbf{x}).$$

Come per il campionamento di Gibbs, dimostrare l'equilibrio dettagliato significa mostrare che il flusso da \mathbf{x} a \mathbf{x}' , $\pi(\mathbf{x})k(\mathbf{x} \rightarrow \mathbf{x}')$, corrisponde al flusso da \mathbf{x}' a \mathbf{x} , $\pi(\mathbf{x}')k(\mathbf{x}' \rightarrow \mathbf{x})$. Una volta sostituita la precedente espressione per $k(\mathbf{x} \rightarrow \mathbf{x}')$, la dimostrazione è piuttosto semplice:

$$\begin{aligned}\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})a(\mathbf{x}'|\mathbf{x}) &= \pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})\min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right) \quad (\text{definizione di } a(\cdot|\cdot)) \\ &= \min(\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x}), \pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x})) \quad (\text{moltiplicando}) \\ &= \pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x})\min\left(\frac{\pi(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}{\pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x})}, 1\right) \quad (\text{dividendo}) \\ &= \pi(\mathbf{x}')q(\mathbf{x}|\mathbf{x})a(\mathbf{x}|\mathbf{x}')\end{aligned}$$

Lasciando da parte le proprietà matematiche, il punto importante del campionamento MH è il rapporto $\pi(\mathbf{x}')/\pi(\mathbf{x})$ nella probabilità di accettazione, che ci dice che, se viene proposto uno stato successivo *più probabile* dello stato corrente, sarà certamente accettato (per ora stiamo ignorando il termine $q(\mathbf{x}|\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})$, che serve a garantire l'equilibrio dettagliato e in molti spazi degli stati è uguale a 1 per simmetria). Se lo stato proposto è *meno probabile* dello stato corrente, la sua probabilità di essere accettato cala proporzionalmente.

Quindi, un criterio per la progettazione di distribuzioni proposte suggerisce di assicurarsi che i nuovi stati proposti siano ragionevolmente probabili. Il campionamento di Gibbs lo fa automaticamente: propone a partire dalla distribuzione di Gibbs quindi la probabilità di generare ogni valore nuovo per X_i è direttamente proporzionale alla sua probabilità (l'Esercizio 13.GIBM vi chiederà di dimostrare che il campionamento di Gibbs è un caso particolare del campionamento MH con probabilità di accettazione uguale a 1).

Un altro criterio è quello di assicurarsi che la catena si “mescoli” bene, il che significa proporre a volte grandi spostamenti verso parti distanti dello spazio degli stati. Nell'esempio precedente, l'uso occasionale di CAMPIONE-PESATO per riavviare la catena in un nuovo stato serve proprio a questo scopo.

Il campionamento MH, oltre a offrire una libertà quasi totale nella progettazione delle distribuzioni proposte, ha altre due proprietà che ne favoriscono l'uso: una è che la probabilità a posteriori $\pi(\mathbf{x}) = P(\mathbf{x} | \mathbf{e})$ appare nel calcolo di accettazione soltanto nella forma di un rapporto $\pi(\mathbf{x}')/\pi(\mathbf{x})$, cosa molto favorevole. Il calcolo diretto di $P(\mathbf{x} | \mathbf{e})$ è proprio quello che cerchiamo di approssimare con il campionamento MH, perciò non avrebbe senso farlo per ogni campione! Utilizziamo invece il seguente trucco:

$$\frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} = \frac{P(\mathbf{x}' | \mathbf{e})}{P(\mathbf{x} | \mathbf{e})} = \frac{P(\mathbf{x}', \mathbf{e})}{P(\mathbf{e})} \frac{P(\mathbf{e})}{P(\mathbf{x}, \mathbf{e})} = \frac{P(\mathbf{x}', \mathbf{e})}{P(\mathbf{x}, \mathbf{e})}.$$

I termini di questo rapporto sono probabilità congiunte complete, cioè prodotti di probabilità condizionate nella rete bayesiana. La seconda proprietà utile di questo rapporto è che, finché la distribuzione proposta apporta soltanto modifiche locali in \mathbf{x} per produrre \mathbf{x}' , soltanto un piccolo numero dei termini del prodotto delle probabilità condizionate sarà diverso. Tutte le probabilità condizionate con variabili a valori invariati si elimineranno nel rapporto. Perciò, come per il campionamento di Gibbs, il lavoro richiesto per generare ogni campione è indipendente dalla dimensione della rete purché i cambiamenti di stato siano locali.

13.4.3 Compilare inferenza approssimata

Gli algoritmi di campionamento delle Figure 13.17, 13.18 e 13.20 condividono una proprietà comune: operano su reti bayesiane rappresentate come strutture dati. Questo sembra del tutto naturale: dopo tutto una rete bayesiana è un grafo aciclico orientato, quindi in quale altro modo la si potrebbe rappresentare? Il problema di questo approccio è che le operazioni richieste per accedere alla struttura dati, per esempio per trovare i genitori di un nodo, sono ripetute migliaia o milioni di volte durante l'esecuzione dell'algoritmo di campionamento, e *tutti questi calcoli non sono affatto necessari*.

La struttura della rete e le probabilità condizionate rimangono fissate per tutto il calcolo, perciò vi è un'opportunità di *compilare* la rete in un codice di inferenza specifico del modello che esegua soltanto i calcoli necessari per l'inferenza su quella rete specifica (se questa idea vi appare familiare, tenete presente che è la stessa usata per la compilazione di programmi logici nel Capitolo 9). Per esempio, supponiamo di voler effettuare un campionamento di Gibbs della variabile *Terremoto* nella rete della Figura 13.2. In base all'algoritmo GIBBS-ASK della Figura 13.20, dobbiamo eseguire il seguente calcolo:

assegnare il valore di *Terremoto* in \mathbf{x} campionandolo da $\mathbf{P}(\text{Terremoto} \mid mb(\text{Terremoto}))$

dove l'ultima distribuzione è calcolata secondo l'Equazione (13.10), che riportiamo di nuovo qui:

$$P(x_i \mid mb(X_i)) = \alpha P(x_i \mid \text{genitori}(X_i)) \prod_{Y_j \in \text{Figli}(X_i)} P(y_j \mid \text{genitori}(Y_j)).$$

Questo calcolo richiede a sua volta di cercare i genitori e i figli di *Terremoto* nella struttura della rete bayesiana; cercare i loro valori correnti; usare tali valori come indici nelle corrispondenti tabelle CPT (anch'esse da trovare nella rete bayesiana); e moltiplicare tra loro tutte le righe appropriate di queste CPT per formare una nuova distribuzione da cui campionare. Infine, come si è osservato nel Paragrafo 13.4.1, lo stesso passo di campionamento deve costruire la versione cumulata della distribuzione discreta e poi trovare il valore che corrisponde a un numero casuale campionato da $[0, 1]$.

Se invece compiliamo la rete, otteniamo un codice di campionamento specifico del modello per la variabile *Terremoto*:

```
r ← un campione casuale uniforme da [0, 1]
if Allarme = true
    then if Intrusione = true
        then return [r < 0,0020212]
        else return [r < 0,36755]
    else if Intrusione = true
        then return [r < 0,0016672]
        else return [r < 0,0014222]
```

Qui le variabili della rete bayesiana *Allarme*, *Intrusione* e così via diventano normali variabili di un programma con valori che comprendono lo stato corrente della catena di Markov. Le espressioni della soglia numerica assumono il valore *true* o *false* e rappresentano le distribuzioni di Gibbs precalcolate per ogni combinazione di valori nella coperta di Markov di *Terremoto*. Il codice non è particolarmente elegante – e tenderà ad allungarsi con l'ampliarsi della rete bayesiana – ma è incredibilmente efficiente. Rispetto al caso di GIBBS-ASK, il codice compilato sarà generalmente più veloce di 2–3 ordini di grandezza; è in grado di svolgere decine di milioni di passi di campionamento al secondo su un normale computer portatile, la sua velocità è limitata per lo più dal costo di generare numeri casuali.

13.5 Reti causali

Abbiamo esaminato diversi vantaggi di mantenere l'ordinamento dei nodi nelle reti bayesiane compatibile con la direzione di causalità. In particolare abbiamo notato quanto sia facile valutare le probabilità condizionate se si mantiene tale ordinamento, e la compattezza della struttura di rete risultante. Abbiamo anche osservato, tuttavia, che in linea di principio con qualsiasi ordinamento dei nodi una rete consistente può essere costruita per rappresentare la funzione di distribuzione congiunta. Questo fatto è stato mostrato chiaramente nella Figura 13.3, in cui cambiando l'ordinamento dei nodi otteniamo reti più fitte e molto meno naturali della rete originale illustrata nella Figura 13.2, che in ogni caso ci consentono di rappresentare la stessa distribuzione su tutte le variabili.

In questo paragrafo trattiamo le **reti causali**, una sottoclasse delle reti bayesiane che escludono tutti gli ordinamenti tranne quelli compatibili con la causalità. Vedremo come costruire tali reti, che vantaggi offrono e come sfruttare tali vantaggi per attività decisionali.

rete causale

Consideriamo la rete bayesiana più semplice che si possa immaginare: una singola freccia *Fuoco* → *Fumo*. Ci indica che le variabili *Fuoco* e *Fumo* potrebbero essere dipendenti, perciò occorre specificare la probabilità a priori $P(\text{Fuoco})$ e la probabilità condizionata $P(\text{Fumo} | \text{Fuoco})$ per specificare la distribuzione congiunta $P(\text{Fuoco}, \text{Fumo})$. Tuttavia, questa distribuzione può essere rappresentata altrettanto bene dalla freccia inversa *Fuoco* ← *Fumo*, usando le appropriate probabilità $P(\text{Fumo})$ e $P(\text{Fuoco} | \text{Fumo})$ calcolate con la regola di Bayes. L'idea che queste due reti siano equivalenti, e quindi contengano le stesse informazioni, suscita disagio e perfino contrarietà nella maggior parte delle persone. Come fanno queste reti a contenere le stesse informazioni se sappiamo che *Fuoco* causa *Fumo* e non viceversa?

In altre parole, sappiamo grazie alla nostra esperienza e alla conoscenza scientifica che eliminare il fumo non fermerebbe il fuoco e che estinguendo il fuoco fermeremmo il fumo. Ci aspettiamo quindi di rappresentare questa assimmetria attraverso la direzione della freccia tra i due nodi. Ma se invertendo la direzione della freccia le cose non cambiano, come possiamo sperare di rappresentare formalmente questa importante informazione?

Le reti bayesiane causali, chiamate anche diagrammi causali, sono state ideate per permetterci di rappresentare assimmetrie causali e di sfruttarle per ragionare con informazioni causali. L'idea è quella di decidere la direzione della freccia sulla base di considerazioni che vanno al di là della dipendenza probabilistica e richiamano un tipo di giudizio del tutto diverso. Anziché chiedere a un esperto se *Fumo* e *Fuoco* sono probabilisticamente dipendenti, come facciamo nelle normali reti bayesiane, ora chiediamo chi reagisce a chi: *Fumo* reagisce a *Fuoco* o *Fuoco* reagisce a *Fumo*?

Questa procedura potrebbe apparire un po' mistica, ma possiamo renderla precisa attraverso il concetto di “assegnamento”, simile all'operatore di assegnamento dei linguaggi di programmazione. Se la natura assegna un valore a *Fumo* sulla base di ciò che apprende su *Fuoco*, tracciamo una freccia da *Fuoco* a *Fumo*. Cosa più importante, se giudichiamo che la natura assegna a *Fuoco* un valore di verità che dipende da altre variabili e non da *Fumo*, evitiamo di tracciare la freccia *Fuoco* ← *Fumo*. In altre parole, il valore x_i di ogni variabile X_i è determinato da un'equazione $x_i = f_i(\text{AltreVariabili})$, e si traccia una freccia $X_j \rightarrow X_i$ se e solo se X_j è uno degli argomenti di f_i .

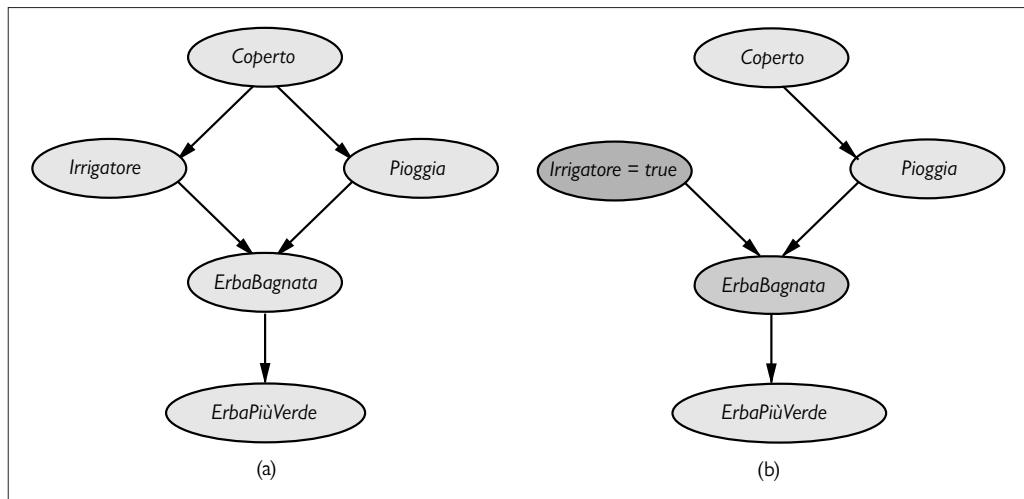
L'equazione $x_i = f_i(\cdot)$ è chiamata **equazione strutturale**, perché descrive un meccanismo stabile in natura che, a differenza delle probabilità che quantificano una rete bayesiana, rimane invariante a misurazioni e cambiamenti locali nell'ambiente.

equazione strutturale

Per apprezzare questa stabilità ai cambiamenti locali considerate la Figura 13.23(a), che illustra una versione leggermente modificata dell'esempio sull'irrigazione dell'erba mostrato nella Figura 13.15. Per rappresentare un irrigatore disabilitato, per esempio, basta eliminare dalla rete tutti i collegamenti che toccano il nodo *Irrigatore*. Per rappresentare un prato coperto da una tenda, eliminiamo la freccia *Pioggia* → *ErbaBagnata*. Qualsiasi riconfigurazione

Figura 13.23

(a) Una rete bayesiana causale che rappresenta relazioni di causa-effetto tra cinque variabili.
 (b) La rete dopo l'esecuzione dell'azione “attiva Irrigatore”.



locale dei meccanismi nell’ambiente può quindi essere traslata, apportando modifiche minori, in una riconfigurazione isomorfa della topologia di rete. Se la rete fosse stata costruita in contrasto con l’ordinamento causale, invece, sarebbe necessaria una trasformazione molto più complessa. Questa stabilità locale è particolarmente importante per la rappresentazione di azioni o interventi, tema che andiamo a discutere nel seguito.

13.5.1 Rappresentare le azioni: l’operatore **do**

Consideriamo ancora l’esempio dell’*Irrigatore* nella Figura 13.23(a). Secondo la semantica standard delle reti bayesiane, la distribuzione congiunta delle cinque variabili è data da un prodotto di cinque distribuzioni condizionate:

$$P(c, p, i, b, v) = P(c) P(p | c) P(i | c) P(b | p, i) P(v | b) \quad (13.14)$$

in cui abbiamo abbreviato i nomi delle variabili (*v* sta per *ErbaPiùVerde*). Il modello in forma di sistema di equazioni strutturali appare come segue:

$$\begin{aligned} C &= f_C(U_C) \\ P &= f_P(C, U_P) \\ I &= f_I(C, U_I) \\ B &= f_B(P, I, U_B) \\ V &= f_V(B, U_V) \end{aligned} \quad (13.15)$$

**variabile non
modellata**

dove, senza perdita di generalità, f_C può essere la funzione identità. Le variabili U in queste equazioni rappresentano **variabili non modellate**, chiamate anche **termini d’errore** o **disturbi**, che perturbano la relazione funzionale tra ogni variabile e i suoi genitori. Per esempio, U_B potrebbe rappresentare un’altra potenziale fonte di umidità, oltre a *Irrigatore* e *Pioggia*, per esempio *RugiadaMattutina* o *ElicotteroPompieri*.

Se tutte le variabili U sono variabili casuali mutuamente indipendenti con probabilità a priori scelte opportunamente, la distribuzione congiunta nell’Equazione (13.14) può essere rappresentata in modo esatto dalle equazioni strutturali dell’Equazione (13.15). Quindi, un sistema di relazioni stocastiche può essere rappresentato da un sistema di relazioni deterministiche, ognuna delle quali è affetta da un disturbo esogeno. Tuttavia, il sistema di equazioni strutturali ci fornisce anche altro: ci consente infatti di predire come gli *interventi* influiranno sull’operatività del sistema e quindi le conseguenze osservabili di tali interventi. Questo non sarebbe possibile utilizzando soltanto la distribuzione congiunta.

Per esempio, supponiamo di attivare l'irrigatore, cioè che noi (che per definizione non siamo parte dei processi causali descritti dal modello) interveniamo a imporre la condizione *Irrigatore = true*. Nella notazione del **do-calculus**, che costituisce una parte fondamentale della teoria delle reti causali, si scrive $do(Irrigatore = true)$. Una volta fatto ciò, la variabile dell'irrigatore non è più dipendente dal fatto che il cielo sia coperto o meno; eliminiamo quindi l'equazione $I = f_I(C, U_I)$ dal sistema di equazioni strutturali e la sostituiamo con $I = \text{true}$, ottenendo:

$$\begin{aligned} C &= f_C(U_C) \\ P &= f_P(C, U_P) \\ I &= \text{true} \\ B &= f_B(P, I, U_B) \\ V &= f_V(B, U_V). \end{aligned} \tag{13.16}$$

do-calculus

Da queste equazioni otteniamo la nuova distribuzione congiunta per le rimanenti variabili condizionate a $do(Irrigatore = \text{true})$:

$$P(c, p, b, v | do(I = \text{true})) = P(c) P(p | c) P(b | p, i = \text{true}) P(v | b) \tag{13.17}$$

Questa corrisponde alla rete “mutilata” nella Figura 13.23(b). Dall'Equazione (13.17) vediamo che le sole variabili le cui probabilità cambiano sono *ErbaBagnata* ed *ErbaPiùVerde*, cioè le discendenti della variabile manipolata *Irrigatore*.

Notate la differenza tra il condizionamento sull'*azione* $do(Irrigatore = \text{true})$ nella rete originale e il condizionamento sull'*osservazione* *Irrigatore = true*. La rete originale ci dice che ci sono meno probabilità che l'irrigatore sia attivo quando il cielo è coperto, perciò se *osserviamo* che l'irrigatore è attivo, ciò riduce la probabilità che il cielo sia coperto. Ma il buon senso ci dice che, se noi (dall'esterno del mondo, per così dire) interveniamo e attiviamo l'irrigatore, questo non influirà sulle condizioni meteorologiche né fornirà nuove informazioni sulle condizioni meteo di quel giorno. Come è mostrato nella Figura 13.23(b), l'intervento interrompe il normale collegamento causale tra le condizioni meteo e l'irrigatore. Così si evita che vi sia qualunque influenza da *Irrigatore* a *Coperto*. Quindi, condizionare su $do(Irrigatore = \text{true})$ nel grafo originale è equivalente a condizionare su *Irrigatore = true* nel grafo mutilato.

Si può adottare un approccio simile per analizzare l'effetto di $do(X_j = x_{jk})$ in una rete causale generale con variabili X_1, \dots, X_n . La rete corrisponde a una distribuzione congiunta definita nel modo consueto (cfr. Equazione (13.2)):

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{genitori}(X_i)). \tag{13.18}$$

Dopo l'applicazione di $do(X_j = x_{jk})$, la nuova distribuzione congiunta omette semplicemente il fattore per X_j :

$$P_{x_{jk}}(x_1, \dots, x_n) = \begin{cases} \prod_{i \neq j} P(x_i | \text{genitori}(X_i)) = \frac{P(x_1, \dots, x_n)}{P(x_j | \text{genitori}(X_j))} & \text{se } x_j = x_{jk} \\ 0 & \text{se } x_j \neq x_{jk} \end{cases} \tag{13.19}$$

Questo segue dal fatto che assegnare a X_j un particolare valore x_{jk} corrisponde a eliminare l'equazione $X_j = f_j(\text{Genitori}(X_j), U_j)$ dal sistema di equazioni strutturali e sostituirla con $X_j = x_{jk}$. Con altre manipolazioni algebriche si può ricavare una formula per l'effetto di assegnare la variabile X_j su ogni altra variabile X_i :

$$\begin{aligned} P(X_i = x_i | do(X_j = x_{jk})) &= P_{x_{jk}}(X_i = x_i) \\ &= \sum_{\text{genitori}(X_i)} P(x_i | x_{jk}, \text{genitori}(X_i)) P(\text{genitori}(X_i)). \end{aligned} \tag{13.20}$$

I termini di probabilità nella somma si ottengono mediante un calcolo nella rete originale, utilizzando un algoritmo di inferenza standard. Questa equazione è nota come **formula di correzione**; è una media pesata sulle probabilità dell'influenza di X_j e dei suoi genitori su X_i , dove i pesi sono le probabilità a priori sui valori dei genitori. Gli effetti di un intervento su più variabili si possono calcolare immaginando una sequenza di interventi singoli, ognuno dei quali elimina le influenze causali su una variabile e porta a un nuovo modello mutilato.

13.5.2 Il criterio back-door

La capacità di predire l'effetto di qualsiasi intervento è un risultato notevole, ma richiede una conoscenza accurata delle distribuzioni condizionate necessarie nel modello, in particolare $P(x_j \mid \text{genitori}(X_j))$. In molte situazioni del mondo reale, tuttavia, tale conoscenza manca. Per esempio, sappiamo che i “fattori genetici” hanno un ruolo nell'obesità, ma non conosciamo quali geni abbiano un ruolo, o la natura precisa dei loro effetti. Anche nel semplice esempio delle decisioni di Mary riguardo l'irrigatore (Figura 13.15, e anche Figura 13.23(a)), potremmo sapere che Mary controlla le condizioni meteo prima di decidere se attivare l'irrigatore, ma non *come* prende la sua decisione.

Il motivo specifico per cui questo aspetto è problematico in questa situazione è che vorremmo predire l'effetto di attivare l'irrigatore su una variabile che segue nell'ordinamento topologico come *ErbaPiùVerde*, ma la formula di correzione (Equazione (13.20)) deve tenere conto non solo del cammino diretto da *Irrigatore*, ma anche del cammino *back-door* (letteralmente “porta sul retro”) che “risale” attraverso *Coperto* e *Pioggia*. Se conoscessimo il valore di *Pioggia*, questo cammino back-door sarebbe bloccato, il che suggerisce che potrebbe esistere un modo di scrivere una formula di correzione che condizioni su *Pioggia* anziché su *Coperto*. E in effetti è possibile:

$$P(g \mid \text{do}(S = \text{true})) = \sum_r P(g \mid S = \text{true}, r) P(r) \quad (13.21)$$

criterio back-door

In generale, se vogliamo trovare l'effetto di $\text{do}(X_j = x_{jk})$ su una variabile X_i , il **criterio back-door** ci consente di scrivere una formula di correzione che condizioni su qualsiasi insieme di variabili \mathbf{Z} che chiuda la back-door, per così dire. In termini più tecnici, ci serve un insieme \mathbf{Z} tale che X_i sia condizionalmente indipendente da $\text{Genitori}(X_j)$ dati X_j e \mathbf{Z} . Questa è un'applicazione diretta della d-separazione (cfr. Paragrafo 13.2.1).

esperimento casuale controllato

Il criterio back-door è un elemento fondamentale per la teoria del ragionamento causale emersa negli ultimi due decenni. Consente di argomentare contro un secolo di dogmi statistici secondo i quali soltanto un **esperimento casuale controllato** poteva fornire informazioni causali. La teoria ha fornito strumenti concettuali e algoritmi per l'analisi causale in un'ampia varietà di situazioni non sperimentali e quasi sperimentali; per calcolare probabilità in affermazioni controfattuali (“se invece fosse accaduto questo, quale sarebbe stata la probabilità?”); per determinare quando sia possibile trasferire i risultati ottenuti in una popolazione a un'altra popolazione; e per gestire tutti i casi in cui mancano dati nell'apprendimento di modelli di probabilità.

13.6 Riepilogo

In questo capitolo abbiamo descritto le **reti bayesiane**, una rappresentazione diffusa e ben sviluppata della conoscenza incerta. Le reti bayesiane hanno un ruolo analogo a quello della logica proposizionale nel trattamento della conoscenza definita.

- Una rete bayesiana è un grafo diretto aciclico i cui nodi corrispondono a variabili casuali; a ogni nodo è associata una distribuzione condizionata, dati i suoi genitori.
- Le reti bayesiane forniscono un modo conciso di rappresentare le relazioni di **indipendenza condizionale** nel dominio.
- Una rete bayesiana specifica una distribuzione di probabilità congiunta sulle sue variabili. La probabilità di ogni assegnamento di tutte le variabili è definita come il prodotto degli elementi corrispondenti nelle distribuzioni condizionate locali. Spesso la rete bayesiana è esponenzialmente più piccola di una distribuzione congiunta enumerata esplicitamente.
- Molte distribuzioni condizionate possono essere rappresentate in modo compatto facendo ricorso a famiglie canoniche di distribuzioni. Le **reti bayesiane ibride**, che includono sia variabili discrete che continue, utilizzano una varietà di distribuzioni canoniche.
- Effettuare una inferenza nelle reti bayesiane significa calcolare la distribuzione di probabilità di un insieme di variabili di query, dato un insieme di variabili di evidenza. Algoritmi di inferenza esatti, come l'**eliminazione di variabili**, valutano somme di prodotti di probabilità condizionate nel modo più efficiente possibile.
- Nei **polialberi** (reti singolarmente connesse), l'inferenza esatta richiede un tempo lineare nelle dimensioni della rete. Nel caso generale il problema è intrattabile.
- Tecniche di campionamento casuale come la **pesatura di verosimiglianza** e gli algoritmi **Monte Carlo per catene di Markov** possono fornire stime ragionevoli delle vere probabilità a posteriori in una rete e riescono a trattare reti molto più grandi di quanto possano fare gli algoritmi esatti.
- Mentre le reti bayesiane catturano inferenze probabilistiche, le **reti causali** catturano relazioni causali e consentono di predire gli effetti degli interventi e delle osservazioni.

Note storiche e bibliografiche

L'uso di reti per rappresentare informazione probabilistica ha preso piede all'inizio del XX secolo, con il lavoro di Sewall Wright sull'analisi probabilistica dell'eredità genetica e i fattori di crescita animale (Wright, 1921, 1934). I. J. Good (1961), in collaborazione con Alan Turing, sviluppò rappresentazioni probabilistiche e metodi di inferenza bayesiana che possono essere considerati i progenitori delle reti bayesiane moderne (benché il suo articolo non sia spesso citato in questo contesto).⁷ Lo stesso lavoro è la fonte originale del modello dell'OR rumoroso.

I **diagrammi di influenza** per la descrizione dei problemi decisionali, che incorporavano una rappresentazione delle variabili casuali mediante grafi diretti aciclici, furono utilizzati nell'analisi delle decisioni alla fine degli anni 1970 (cfr. Capitolo 16): tuttavia, per la

loro valutazione si usava solo l'enumerazione. Judea Pearl sviluppò il metodo per l'inferenza nelle reti ad albero (Pearl, 1982a) e polialbero (Kim e Pearl, 1983) e spiegò l'importanza dei modelli di probabilità causali anziché diagnostici. Il primo sistema esperto che sfruttava le reti bayesiane fu CONVINCE (Kim, 1983).

Come si è visto nel Capitolo 1, a metà degli anni 1980 c'è stato un boom dei sistemi esperti basati su regole, che incorporavano metodi *ad hoc* per gestire

⁷ I. J. Good era lo statistico capo del gruppo guidato da Turing che, durante la Seconda Guerra Mondiale, aveva il compito di decodificare i codici nemici. In 2001: *Odissea nello Spazio* (Clarke, 1968a), a Good e Minsky vengono accreditate le scoperte rivoluzionarie che avevano portato allo sviluppo del computer HAL 9000.

l'incertezza. La probabilità veniva considerata impraticabile e “cognitivamente non plausibile” come base per il ragionamento. Il combattivo articolo di Peter Cheeseman (1985) “In Defense of Probability” e l'altro suo articolo “An Inquiry into Computer Understanding” (Cheeseman, 1988, con commenti) contribuirono a cambiare le cose.

Il ritorno dell'interesse verso la probabilità, tuttavia, si deve principalmente a Pearl con lo sviluppo delle reti bayesiane e di un approccio probabilistico all'IA, come descritto nel suo libro *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). Il libro trattava sia problemi di rappresentazione, incluse le relazioni di indipendenza condizionale e il criterio di d-separazione, sia approcci algoritmici. Geiger *et al.* (1990a) e Tian *et al.* (1998) presentarono fondamentali risultati computazionali sull'individuazione efficiente della d-separazione.

Eugene Charniak aiutò a presentare le idee di Pearl ai ricercatori di IA con un articolo molto popolare, “Bayesian networks without tears”⁸ (1991), e un libro (1993). Anche il libro di Dean e Wellman (1991) contribuì a introdurre i ricercatori di IA alle reti bayesiane. Shachter (1998) presentò un metodo semplificato per determinare la d-separazione, denominato algoritmo “Bayes-ball”.

Con lo sviluppo di applicazioni delle reti bayesiane, i ricercatori ritennero necessario andare oltre il modello di base delle variabili discrete con CPT. Per esempio, il sistema CPCS (Pradhan *et al.*, 1994), una rete bayesiana per la medicina interna con 448 nodi e 906 collegamenti, faceva ampio uso degli operatori logici rumorosi proposti da Good (1961). Boutilier *et al.* (1996) analizzarono i vantaggi algoritmici dell'indipendenza specifica del contesto. L'inclusione di variabili casuali continue in reti bayesiane fu presa in considerazione da Pearl (1988) e da Shachter e Kenley (1989), con articoli in cui si discuteva di reti contenenti soltanto variabili continue con distribuzioni gaussiane lineari.

Le reti ibride con variabili discrete e continue furono studiate da Lauritzen e Wermuth (1989) e implementate nel sistema CHUGIN (Olesen, 1993). Ulteriori analisi di modelli gaussiani lineari, con connessioni a molti altri modelli usati in statistica, si trovano in Roweis e Ghahramani (1999); Lerner (2002) fornì una trattazione molto approfondita dell'uso di tali modelli in reti bayesiane ibride. La distribuzione probit viene solitamente attribuita a Gaddum (1933) e Bliss (1934), anche se è stata scoperta più volte nel diciannovesimo secolo. Il lavoro di Bliss fu ampliato notevolmente da Finney (1947). La distribuzione probit

è stata ampiamente usata per la modellazione di fenomeni a scelta discreta e può essere estesa per gestire più di due scelte (Daganzo, 1979). Il modello expit (logit inverso) fu introdotto da Berkson (1944); inizialmente deriso, alla fine divenne più popolare del modello probit. Bishop (1995) fornisce una motivazione semplice per il suo utilizzo.

Tra le prime applicazioni delle reti bayesiane in medicina vi è il sistema MUNIN per la diagnosi dei disturbi neuromuscolari (Andersen *et al.*, 1989) e del sistema PATHFINDER per la patologia (Heckerman, 1991). Tra le applicazioni in ingegneria vi sono il lavoro dell'Electric Power Research Institute sul monitoraggio dei generatori di potenza (Morjaria *et al.*, 1995), il lavoro della NASA sulla visualizzazione di informazioni temporalmente critiche presso il centro di controllo delle missioni a Houston (Horvitz e Barry, 1995), e il campo generale della **tomografia di rete**, che punta a inferire proprietà locali non osservate di nodi e collegamenti di Internet da osservazioni delle prestazioni nel trasferimento di messaggi *end-to-end* (Castro *et al.*, 2004). Tra i sistemi basati su reti bayesiane di più largo utilizzo vi sono i moduli di diagnostica e riparazione (per esempio per la procedura di stampa) integrati in Microsoft Windows (Breese e Heckerman, 1996) e l'Assistente di Office in Microsoft Office (Horvitz *et al.*, 1998).

Un altro importante campo di applicazione è la biologia: i modelli matematici usati per analizzare l'ereditarietà genetica negli alberi genealogici (la cosiddetta **analisi del pedigree**) sono in effetti una particolare forma di reti bayesiane. Algoritmi di inferenza esatta per l'analisi del pedigree, simili alle tecniche di eliminazione di variabili, furono sviluppati negli anni 1970 (Cannings *et al.*, 1978). Le reti bayesiane sono state usate per identificare i geni umani facendo riferimento ai geni del topo (Zhang *et al.*, 2003), per inferire reti cellulari (Friedman, 2004), per l'analisi di collegamenti genetici per localizzare geni correlati a malattie (Silberstein *et al.*, 2013) e per molte altre attività nel campo della bioinformatica. Potremmo proseguire citando altre applicazioni, ma preferiamo rimandare a Pourret *et al.* (2008), una guida di 400 pagine alle applicazioni delle reti bayesiane. Solo nell'ultimo decennio le applicazioni rese pubbliche sono decine di migliaia, spaziando dall'odontoiatria ai modelli climatici.

⁸ Il titolo dell'articolo nella versione originale era “Pearl for swine” (con un gioco di parole sul cognome Pearl che in inglese significa perla, quindi “Perla per maiali”).

Judea Pearl (1985), nel primo articolo in cui fu utilizzato il termine “reti bayesiane” descrisse brevemente un algoritmo di inferenza per reti generali basato sull’idea di condizionamento con insieme di taglio descritta nel Capitolo 6. In modo indipendente, Ross Shachter (1986), lavorando nella comunità dei diagrammi di influenza, sviluppò un algoritmo completo basato sulla riduzione della rete guidata da un obiettivo, mediante trasformazioni che preservano le probabilità a posteriori.

Pearl (1986) sviluppò un algoritmo di clustering per l’inferenza esatta in reti bayesiane generali, passando per una conversione in polialberi orientati di cluster in cui si sfruttava il passaggio di messaggi per assicurare la consistenza delle variabili condivise. Un approccio analogo, sviluppato dagli statistici David Spiegelhalter e Steffen Lauritzen (Lauritzen e Spiegelhalter, 1988), è basato sulla conversione in una forma non orientata di un modello grafico chiamato **rete di Markov**. Questo approccio è implementato nel sistema HUGIN, uno strumento efficiente e largamente diffuso per il ragionamento incerto (Andersen *et al.*, 1989).

L’idea di base dell’eliminazione delle variabili – per cui si possono evitare i calcoli ripetuti nell’espressione di somma di prodotti utilizzando un sistema di caching – apparve nell’algoritmo di inferenza probabilistica simbolica (SPI) (Shachter *et al.*, 1990). L’algoritmo di eliminazione descritto in questo libro è più vicino a quello sviluppato da Zhang e Poole (1994). I criteri per potare le variabili irrilevanti furono sviluppati da Geiger *et al.* (1990b) e da Lauritzen *et al.* (1990); il criterio da noi fornito è un semplice caso speciale di questi. Dechter (1999) mostrò come l’idea dell’eliminazione delle variabili sia sostanzialmente identica alla **programmazione dinamica non seriale** (Bertele e Brioschi, 1972).

Questo risultato collega gli algoritmi sulle reti bayesiane ai metodi per risolvere i CSP e ci fornisce una misura diretta della complessità dell’inferenza esatta in rapporto alla **larghezza d’albero** della rete. Per evitare la crescita esponenziale della dimensione dei fattori nell’eliminazione di variabili si possono scartare variabili dai fattori grandi (Dechter e Rish, 2003); è anche possibile limitare l’errore introdotto in questo modo (Wexler e Meek, 2009). In alternativa, è possibile comprimere i fattori rappresentandoli mediante diagrammi decisionali anziché mediante tabelle (Goate e Domingos, 2011).

Tra i metodi esatti basati sull’enumerazione ricorsiva (Figura 13.11) combinati con il caching vi sono

l’algoritmo di condizionamento ricorsivo (Darwiche, 2001), l’algoritmo di eliminazione dei valori (Bacchus *et al.*, 2003) e la ricerca AND–OR (Dechter e Mateescu, 2007). Il metodo del conteggio con modello ponderato (Sang *et al.*, 2005; Chavira e Darwiche, 2008) è solitamente basato su un risolutore SAT in stile DPLL (cfr. Figura 7.17 nel Capitolo 7); come tale, esegue anch’esso un’enumerazione ricorsiva degli assegnamenti di variabile con caching, perciò l’approccio è piuttosto simile. Tutti e tre questi algoritmi possono implementare ogni varietà di compromessi tra spazio e tempo. Poiché considerano assegnazioni di variabili, questi algoritmi possono facilmente sfruttare il determinismo e l’indipendenza specifica del contesto nel modello. Possono anche essere modificati per utilizzare un algoritmo efficiente in tempo lineare, ogni volta che l’assegnamento parziale fa diventare la rete rimanente un polialbero (questa è una versione del metodo di **condizionamento con insieme di taglio** descritto nel Capitolo 6 per i CSP). Per l’inferenza esatta in modelli di grandi dimensioni, in cui i requisiti di spazio per il clustering e l’eliminazione di variabili diventano enormi, questi algoritmi ricorsivi spesso offrono l’approccio più pratico.

Oltre al calcolo di probabilità marginali, il campo delle reti bayesiane comprende altre importanti attività di inferenza. La **spiegazione più probabile** o MPE (*most probable explanation*) è l’assegnamento più probabile a variabili non di evidenza data l’evidenza (l’inferenza MPE è un caso speciale dell’inferenza MAP – *maximum a posteriori* – in cui l’assegnamento più probabile deve essere un *sottoinsieme* di variabili non di evidenza data l’evidenza). Per problemi di questo tipo sono stati sviluppati numerosi algoritmi, alcuni legati alla ricerca del cammino più breve e alla ricerca AND–OR; un riepilogo è fornito in Marinescu e Dechter (2009).

Il primo risultato sulla complessità dell’inferenza in reti bayesiane si deve a Cooper (1990), che dimostrò che il problema generale di calcolare probabilità marginali in reti bayesiane è NP-difficile; come si è osservato in questo capitolo, questo risultato si può rafforzare alla complessità #P-difficile attraverso una riduzione dal conteggio degli assegnamenti soddisfacenti (Roth, 1996). Questo implica che anche l’inferenza approssimata è NP-difficile (Dagum e Luby, 1993); tuttavia, nel caso in cui le probabilità possano essere escluse dagli estremi 0 e 1, una forma di pesatura di verosimiglianza converge in tempo polinomiale (randomizzato) (Dagum e Luby, 1997). Shimony (1994) ha dimostrato che il problema di trovare la

spiegazione più probabile è NP-completo – intrattabile, ma in qualche modo più facile di calcolare le probabilità marginali – mentre Park e Darwiche (2004) hanno fornito un'estesa analisi del calcolo MAP, dimostrando che rientra nella classe dei problemi NP^{PP}-completi – quindi è più difficile del calcolo delle probabilità marginali.

Lo sviluppo di algoritmi approssimati veloci per l'inferenza sulle reti bayesiane è un'area molto attiva, a cui contribuisce la statistica, l'informatica e la fisica. Il metodo del campionamento di rigetto è una tecnica generale che risale almeno all'ago di Buffon (1777); è stato applicato per la prima volta alle reti bayesiane da Max Henrion (1988), che lo chiamò **campionamento logico**. Il campionamento di importanza fu inventato per applicazioni in fisica (Kahn, 1950a, 1950b) e applicato all'inferenza in reti bayesiane da Fung e Chang (1989) (che chiamarono l'algoritmo “pesatura basata sull'evidenza”) e da Shachter e Peot (1989).

In statistica, il **campionamento adattativo** è stato applicato a ogni tipo di algoritmi Monte Carlo per velocizzare la convergenza. L'idea di base è di adattare la distribuzione da cui sono generati i campioni sulla base dei risultati ottenuti da campioni precedenti. Gilks e Wild (1992) svilupparono il campionamento di rigetto adattativo, mentre il campionamento di importanza adattativo sembra essere stato sviluppato in modo indipendente nei campi della fisica (Lepage, 1978), dell'ingegneria civile (Karamchandani *et al.*, 1989), della statistica (Oh e Berger, 1992), e della grafica digitale (Veach e Guibas, 1995). Cheng e Druzdzel (2000) hanno descritto una versione adattativa del campionamento di importanza applicata all'inferenza in reti bayesiane. Più recentemente, Le *et al.* (2017) hanno illustrato l'uso di sistemi di deep learning per produrre distribuzioni proposte in grado di velocizzare il campionamento di importanza di molti ordini di grandezza.

Gli algoritmi Monte Carlo per catene di Markov (MCMC) hanno avuto inizio con l'algoritmo di Metropolis, dovuto a Metropolis *et al.* (1953), che è anche la fonte del simulated annealing descritto nel Capitolo 4. Hastings (1970) introdusse il passo di accettazione/rifiuto che costituisce parte integrante di ciò che chiamiamo algoritmo di Metropolis-Hastings. Il campionatore di Gibbs fu inventato da Geman e Geman (1984) per l'inferenza nelle reti di Markov non orientate. L'applicazione del campionamento di Gibbs alle reti bayesiane si deve a Pearl (1987). Gli articoli raccolti da Gilks *et al.* (1996) trattano la teoria e le applicazioni degli algoritmi MCMC.

Fin dalla metà degli anni 1990, gli algoritmi MCMC sono diventati la colonna portante della statistica bayesiana e del calcolo statistico in molte altre discipline, tra cui fisica e biologia. L'*Handbook of Markov Chain Monte Carlo* (Brooks *et al.*, 2011) tratta molti aspetti della letteratura su questo argomento. Il pacchetto BUGS (Gilks *et al.*, 1994) fu uno dei primi e più influenti sistemi per la modellazione di reti bayesiane e l'inferenza con campionamento di Gibbs. STAN (da Stanislaw Ulam, uno dei primi a utilizzare metodi Monte Carlo in fisica) è un sistema più recente che fa uso di inferenza Monte Carlo hamiltoniana (Carpenter *et al.*, 2017).

Ci sono due famiglie molto importanti di metodi approssimati che non abbiamo trattato nel capitolo. La prima è quella dei metodi di **approssimazione variazionale**, che possono aiutare a semplificare calcoli complessi di ogni genere. L'idea base è proporre una versione ridotta del problema originale molto simile a esso, ma più semplice da trattare. Il problema ridotto è descritto da alcuni **parametri variazionali** λ regolati per minimizzare una funzione di distanza D tra il problema originale e quello ridotto, spesso risolvendo il sistema di equazioni $\delta D / \delta \lambda = 0$. In molti casi si possono ottenere limiti superiori e inferiori precisi. I metodi variazionali sono stati usati per lungo tempo in campo statistico (Rustagi, 1976). Nella fisica statistica, il metodo del **campo medio** è una particolare approssimazione variazionale in cui si assume che le singole variabili che compongono il modello siano completamente indipendenti. Questa idea è stata applicata alla risoluzione di grandi reti di Markov non orientate (Peterson e Anderson, 1987; Parisi, 1988). Saul *et al.* (1996) hanno sviluppato i fondamenti matematici per l'applicazione dei metodi variazionali alle reti bayesiane e hanno calcolato, sfruttando i metodi di campo medio, accurate approssimazioni limitate inferiormente per reti a sigmoide. Jaakkola e Jordan (1996) hanno esteso la metodologia per ottenere sia limiti superiori che inferiori. Dopo questi primi articoli, i metodi variazionali sono stati applicati a molte famiglie specifiche di modelli. L'importante articolo di Wainwright e Jordan (2008) ha presentato un'analisi teorica unificante della letteratura sull'argomento.

Una seconda famiglia importante di algoritmi di approssimazione deriva da quello di Pearl basato sul passaggio di messaggi in un polialbero (1982a). Quest'algoritmo si può applicare a reti generiche “cicliche”, come suggerito dallo stesso Pearl (1988). I risultati potrebbero essere errati, e l'algoritmo potrebbe anche

non terminare, ma in molti casi i valori ottenuti sono vicini a quelli reali. L'approccio della cosiddetta **propagazione ciclica delle credenze** non ha ricevuto molta attenzione finché McEliece *et al.* (1998) non ha osservato che è esattamente il calcolo eseguito dall'algoritmo **turbo decoding** (Berrou *et al.*, 1993), che ha rappresentato un passo avanti decisivo nella progettazione di codici efficienti a correzione d'errore.

Queste osservazioni implicano che la propagazione ciclica delle credenze è sia veloce che accurata sulle reti molto grandi e fittamente interconnesse usate per la decodifica, ragion per cui potrebbe risultare utile per un uso più generale. Supporto teorico per questi risultati, incluse dimostrazioni di convergenza per alcuni casi speciali, è stato fornito da Weiss (2000b), Weiss e Freeman (2001), Yedidia *et al.* (2005), sulla base di connessioni con concetti della fisica statistica.

Teorie dell'inferenza causale che vanno oltre gli esperimenti casuali controllati furono proposte da Rubin (1974) e Robins (1986), ma queste idee sono rimaste oscure e controverse fino a quando Judea Pearl sviluppò e presentò una teoria completa e articolata della causalità basata sulle reti causali (Pearl, 2000). Peters *et al.* (2017) hanno ulteriormente sviluppato la teoria, focalizzandosi in particolare sull'apprendimento. Un lavoro più recente, *The Book of Why* (Pearl e McKenzie, 2018), fornisce un'introduzione meno matematica ma più leggibile e ad ampio spettro.

Il ragionamento incerto nell'IA non è sempre stato basato sulla teoria delle probabilità. Come si è notato nel Capitolo 12, i primi sistemi probabilistici furono messi nell'ombra durante i primi anni 1970, lasciando un vuoto destinato a essere riempito da metodi alternativi, tra cui i sistemi esperti basati su regole, la teoria di Dempster–Shafer theory e (in una certa misura) la logica fuzzy.⁹

Gli approcci all'incertezza basati sulle regole speravano di sfruttare il successo dei sistemi logici basati su regole, aggiungendo a ogni regola una sorta di "fattore di confusione" – chiamato con un termine politicamente corretto **fattore di certezza** – per lasciare spazio all'incertezza. Il primo sistema di quel tipo fu MYCIN (Shortliffe, 1976), un sistema esperto medico per le infezioni batteriche. La raccolta *Rule-Based Expert Systems* (Buchanan e Shortliffe, 1984) fornisce una panoramica completa su MYCIN e i suoi discendenti (vedi anche Stefik, 1995).

David Heckerman (1986) mostrò che una versione leggermente modificata dei calcoli con fattore di certezza fornisce risultati probabilistici corretti in alcuni casi, ma in altri compie gravi errori di sovrastima delle

evidenze. Quando gli insiemi di regole diventavano grandi, si facevano più comuni interazioni non desiderabili tra le regole, e i ricercatori determinarono che i fattori di certezza di molte altre regole dovevano essere "messi a punto" quando si aggiungevano nuove regole. Le proprietà matematiche di base che consentono le *catene* di ragionamento nella logica non valgono per la probabilità.

La teoria di Dempster–Shafer ha avuto origine con un articolo di Arthur Dempster (1968) che proponeva di generalizzare la probabilità introducendo intervalli di valori e una regola di combinazione per usarli. Un tale approccio potrebbe alleviare la difficoltà di specificare le probabilità in modo esatto. Il lavoro successivo di Glenn Shafer (1976) portò a vedere la teoria di Dempster–Shafer come un approccio alternativo a quello probabilistico. Pearl (1988) e Ruspini *et al.* (1992) analizzarono le relazioni tra la teoria di Dempster–Shafer e la teoria delle probabilità standard. In molti casi, la teoria delle probabilità non richiede che le probabilità siano specificate in modo esatto: si può esprimere incertezza sui valori di probabilità come distribuzioni di probabilità (del secondo ordine), come è spiegato nel Capitolo 20 del Volume 2.

Gli **insiemi fuzzy** furono sviluppati da Lotfi Zadeh (1965) per rispondere alle difficoltà riscontrate nel fornire input esatti ai sistemi intelligenti. Un insieme fuzzy è un insieme in cui vi sono vari gradi di appartenenza. La **logica fuzzy** è un metodo per ragionare con espressioni logiche che descrivono l'appartenenza a insiemi fuzzy. Il **controllo fuzzy** è una metodologia per costruire sistemi di controllo in cui la corrispondenza tra input a valori reali e parametri di output è rappresentata da regole fuzzy. Questa metodologia ha ottenuto grande successo in prodotti commerciali come trasmissioni automatiche, videocamere e rasoi elettrici. Il testo di Zimmermann (2001) fornisce un'introduzione particolareggiata alla teoria degli insiemi fuzzy; molti articoli sulle applicazioni fuzzy sono raccolti in Zimmermann (1999).

La logica fuzzy è stata spesso percepita – sbagliando – come concorrente diretta della teoria delle probabilità, mentre in effetti affronta problematiche diverse: anziché considerare l'incertezza sulla verità di proposizioni ben definite, la logica fuzzy affronta la

⁹ Un quarto approccio, il **ragionamento di default**, considera le conclusioni non come "credete fino a un certo livello", ma come "credete finché non si trovi un motivo per credere altro". L'argomento è trattato nel Capitolo 10.

vaghezza nella corrispondenza dai termini di una teoria simbolica a un mondo reale. La vaghezza è una problematica reale in qualsiasi applicazione alla realtà della logica, della probabilità e anche dei modelli matematici standard. Anche una variabile impeccabile come la massa del pianeta Terra risulta, a un esame attento, variare con il tempo, con l'arrivo di meteoriti e varie molecole che entrano ed escono. È anche imprecisa: comprende l'atmosfera? Se sì, fino a quale altezza? In alcuni casi, un'ulteriore elaborazione del modello è in grado di ridurre la vaghezza, ma la logica fuzzy considera la vaghezza come un fatto scontato e sviluppa una teoria per affrontarla.

La **teoria della possibilità** (Zadeh, 1978) fu introdotta per gestire l'incertezza nei sistemi fuzzy e ha molti punti in comune con la probabilità (Dubois e Prade, 1994).

Molti ricercatori in IA negli anni 1970 rifiutavano la teoria delle probabilità perché i calcoli numerici richiesti non apparivano evidenti all'introspezione e presumevano un livello di precisione della nostra conoscenza incerta non realistico. Lo sviluppo delle **reti probabilistiche qualitative** (Wellman, 1990a) fornì un'astrazione puramente qualitativa delle reti bayesiane, usando solo la nozione di influenza positiva e

negativa tra variabili. Wellman dimostra che, in molti casi, informazioni siffatte sono sufficienti per arrivare a una decisione ottima senza la necessità di specificare precisamente i valori di probabilità. Goldszmidt e Pearl (1996) adottarono un approccio simile. Il lavoro di Darwiche e Ginsberg (1992) estrapolò dalla teoria della probabilità le caratteristiche fondamentali del condizionamento e della combinazione di evidenze e mostrò che possono essere applicate anche nel ragionamento logico e di default.

Diversi testi eccellenti (Jensen, 2007; Darwiche, 2009; Koller e Friedman, 2009; Korb e Nicholson, 2010; Dechter, 2019) presentano una trattazione completa degli argomenti di questo capitolo. I nuovi sviluppi sul ragionamento probabilistico compaiono sia in riviste di IA come *Artificial Intelligence* e il *Journal of AI Research* che in pubblicazioni più specializzate, come l'*International Journal of Approximate Reasoning*. Nelle riviste di statistica compaiono molti articoli sui modelli a grafo, che includono le reti bayesiane. Gli atti di congressi come Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NeurIPS) e Artificial Intelligence and Statistics (AISTATS) sono ottime fonti per approfondire gli ultimi sviluppi della ricerca.

CAPITOLO

14

Ragionamento probabilistico nel tempo

- 14.1 Tempo e incertezza
- 14.2 Inferenza nei modelli temporali
- 14.3 Modelli di Markov nascosti
- 14.4 Filtri di Kalman
- 14.5 Reti bayesiane dinamiche
- 14.6 Riepilogo
 - Note storiche e bibliografiche

In cui tentiamo di interpretare il presente, comprendere il passato e forse predire il futuro, anche quando di chiaro c'è ben poco.

Gli agenti che operano in ambienti parzialmente osservabili devono essere in grado di tenere traccia dello stato corrente, almeno per quanto consentono i loro sensori. Nel Paragrafo 4.4 abbiamo illustrato una metodologia per fare ciò: un agente mantiene uno **stato-credenza** che rappresenta quali stati del mondo sono attualmente possibili; con questo stato-credenza e con un **modello di transizione**, l'agente è in grado di predire come il mondo potrebbe evolvere al passo temporale successivo. Lo stato-credenza può essere aggiornato dall'agente a partire dalle percezioni osservate e da un **modello sensoriale**. Questo è un concetto pervasivo: nel Capitolo 4 gli stati-credenza sono rappresentati da insiemi di stati enumerati esplicitamente, mentre nei Capitoli 7 e 11 sono rappresentati da formule logiche. Questi approcci definiscono gli stati-credenza in termini di quali mondi sono *possibili*, ma non possono dire nulla su quali stati sono *probabili* o *improbabili*; in questo capitolo utilizzeremo la teoria delle probabilità per quantificare il grado di credenza in elementi dello stato-credenza.

Come mostriamo nel Paragrafo 14.1, il tempo viene gestito esattamente come nel Capitolo 7: per modellare un mondo mutevole si usa una variabile per ogni aspetto del suo stato *in ogni istante temporale*. Il modello di transizione e il modello sensoriale potrebbero essere incerti: il modello di transizione descrive la distribuzione di probabilità delle variabili al tempo t dato lo stato del mondo in tempi passati, mentre il modello sensoriale descrive la probabilità di ogni percezione al tempo t dato lo stato corrente del mondo. Il Paragrafo 14.2 definisce gli aspetti fondamentali dell'inferenza nei modelli temporali e illustra la struttura generale degli algoritmi. Sono poi descritti tre tipi specifici di modelli: **modelli di Markov nascosti**, **filtri di Kalman** e **reti bayesiane dinamiche** (che includono i due precedenti come casi speciali).

14.1 Tempo e incertezza

Abbiamo sviluppato le nostre tecniche per il ragionamento probabilistico nel contesto di mondi *statici*, in cui ogni variabile ha un singolo valore fisso. Per esempio, quando cerchiamo di riparare un’automobile presumiamo che qualsiasi cosa sia rotta rimanga rotta per tutta la durata del processo di diagnosi; il nostro compito è inferire lo stato della macchina dalle evidenze osservate, anch’esse immutabili.

Ora considerate un problema leggermente diverso: il trattamento di un paziente diabetico. Come nel caso della riparazione meccanica, abbiamo a disposizione delle evidenze, quali le dosi di insulina assunte recentemente, la dieta, il tasso di zucchero nel sangue e altri sintomi fisici. Il compito è valutare lo stato corrente del paziente, incluso l’effettivo livello glicemico e quello dell’insulina nel sangue. Date queste informazioni, possiamo prendere una decisione riguardo all’assunzione di cibo e alla dose di insulina. A differenza del caso dell’automobile, ora gli aspetti *dinamici* del problema sono essenziali. I livelli di zucchero nel sangue e le relative misurazioni possono cambiare rapidamente nel tempo, a seconda dell’assunzione recente di cibo e insulina, dell’attività metabolica, dell’ora del giorno e così via. Per valutare lo stato corrente partendo dalla storia delle evidenze e prevedere l’esito delle azioni terapeutiche, è necessario modellare tali cambiamenti.

Le stesse considerazioni sorgono in molti altri contesti, come il tracciamento della posizione di un robot, l’analisi dell’attività economica di una nazione, la comprensione della sequenza di parole pronunciate o scritte da una persona. Come è possibile modellare situazioni dinamiche come queste?

14.1.1 Stati e osservazioni

tempo discreto
time slice

In questo capitolo esaminiamo modelli a **tempo discreto**, in cui il mondo è considerato in una serie di istantanee o **time slice** (letteralmente “fette temporali”)¹ che numereremo con 0, 1, 2 e così via, anziché assegnarvi tempi specifici. Generalmente si assume che l’intervallo Δ tra le time slice sia lo stesso per ogni intervallo. Per ogni singola applicazione è necessario scegliere un valore specifico di Δ . A volte la scelta è determinata dal sensore; per esempio, una videocamera potrebbe fornire immagini a intervalli di 1/30 di secondo. In altri casi l’intervallo è determinato dalla velocità di variazione di variabili rilevanti; per esempio, nel caso del monitoraggio del livello di glucosio nel sangue, la situazione può cambiare notevolmente in dieci minuti, perciò potrebbe risultare appropriato scegliere un intervallo di un minuto. Invece, nella modellazione della deriva dei continenti in tempo geologico, potrebbe andare bene un intervallo di milioni di anni.

Ogni time slice in un modello di probabilità a tempo discreto contiene un insieme di variabili casuali, alcune osservabili e altre no. Per semplicità assumeremo che in ogni time slice sia osservabile lo stesso sottoinsieme di variabili, benché ciò non sia strettamente necessario per la nostra trattazione. Denoteremo con \mathbf{X}_t l’insieme delle variabili di stato al tempo t , che si assume siano non osservabili, e con \mathbf{E}_t l’insieme delle variabili di evidenza osservabili. L’osservazione al tempo t sarà $\mathbf{E}_t = \mathbf{e}_t$ per qualche insieme di valori \mathbf{e}_t .

Considerate il seguente esempio: supponete di essere la guardia di sicurezza di un’installazione sotterranea segreta. Volete sapere se oggi sta piovendo, ma il vostro unico contatto con il mondo esterno avviene ogni mattina, quando vedete arrivare il direttore con o senza ombrello. Per ogni giorno t , l’insieme \mathbf{E}_t contiene così una sola variabile di evidenza denominata *Ombrello* o U_t (presenza o meno dell’ombrello), e l’insieme \mathbf{X}_t contiene una singola va-

¹ L’incertezza in *tempo continuo* può essere modellata mediante **equazioni differenziali stocastiche** (SDE, *stochastic differential equations*). I modelli studiati in questo capitolo possono essere visti come approssimazioni di SDE in tempo discreto.

riabile di stato *Pioggia* o R_t (sta piovendo o no). Altri problemi possono coinvolgere un numero più ampio di variabili. Nell'esempio del diabete le variabili di evidenza potrebbero essere *GlicemiaMisurata_t* e *PulsazioniCardiache_t*, mentre le variabili di stato potrebbero essere *Glicemia_t* e *ContenutoDelloStomaco_t* (notate che *GlicemiaMisurata_t* e *Glicemia_t* non sono affatto la stessa variabile; è proprio in questo modo che si gestiscono le misurazioni rumorose).

Ipotizzeremo che la sequenza degli stati inizi in $t = 0$ e che le evidenze comincino ad arrivare nell'istante $t = 1$. Di conseguenza, il nostro mondo dell'ombrellino sarà rappresentato dalle variabili di stato R_0, R_1, R_2, \dots e da quelle di evidenza U_1, U_2, \dots . Useremo la notazione $a:b$ per indicare la sequenza di interi da a a b (inclusi) e $\mathbf{X}_{a:b}$ per denotare l'insieme di variabili da \mathbf{X}_a a \mathbf{X}_b inclusi. Per esempio, $U_{1:3}$ corrisponde a U_1, U_2, U_3 (notate che questa notazione è diversa da quella usata in linguaggi di programmazione come Python e Go, in cui $U[1, 3]$ non include $U[3]$).

14.1.2 Modelli di transizione e modelli sensoriali

Avendo deciso l'insieme di variabili di stato e di evidenza per un dato problema, il passo successivo è specificare come si evolve il mondo (il modello di transizione) e come le variabili di evidenza ottengono i loro valori (il modello sensoriale).

Il modello di transizione specifica la distribuzione di probabilità sulle ultime variabili di stato dati i valori precedenti, cioè $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1})$. Ora incontriamo un problema: l'insieme $\mathbf{X}_{0:t-1}$ ha dimensione illimitata al crescere di t . Risolviamo questo problema con una **ipotesi di Markov**: che lo stato corrente dipenda solo da un *numero fissato e finito* di stati precedenti. I processi che soddisfano questo requisito sono stati studiati approfonditamente per la prima volta dallo statistico Andrei Markov (1856–1922) e si chiamano **processi di Markov** o **catene di Markov**. Ne esistono vari tipi: il più semplice è il **processo di Markov del primo ordine**, in cui lo stato corrente dipende solo da quello immediatamente precedente e nessun altro. In altre parole, uno stato fornisce informazione sufficiente per rendere il futuro condizionalmente indipendente dal passato, e abbiamo:

$$\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1}). \quad (14.1)$$

Di conseguenza, in un processo di Markov del primo ordine il modello di transizione è la distribuzione condizionata $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$. Il modello di transizione per un processo di Markov del secondo ordine è la distribuzione condizionata $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$. La Figura 14.1 mostra le strutture delle reti bayesiane corrispondenti ai processi di Markov del primo e secondo ordine.

Anche con l'ipotesi di Markov, rimane un problema: esistono infiniti valori possibili di t . Dobbiamo specificare una diversa distribuzione per ogni passo temporale? Evitiamo questo problema ipotizzando che le modifiche nello stato del mondo siano causate da un processo **omogeneo rispetto al tempo**, cioè governato da leggi che non cambiano nel tempo. Nel mondo dell'ombrellino, quindi, la probabilità condizionata che piova, $\mathbf{P}(R_t | R_{t-1})$, è la stessa per tutti i valori t , quindi dobbiamo specificare una sola tabella delle probabilità condizionate.

ipotesi di Markov

processo di Markov

processo di Markov del primo ordine

omogeneo rispetto al tempo

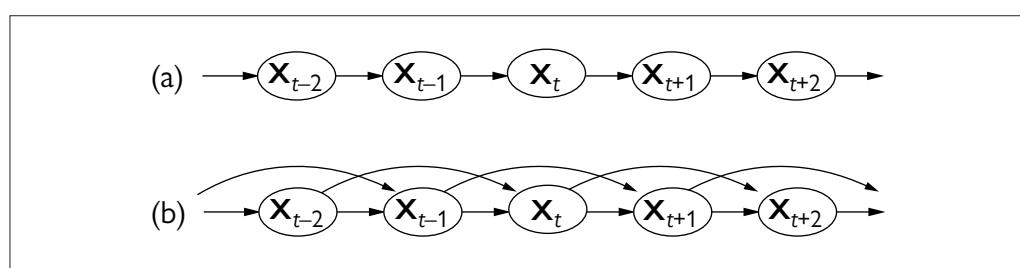


Figura 14.1 (a) Struttura della rete bayesiana corrispondente a un processo di Markov del primo ordine con stato definito dalle variabili \mathbf{X}_t . (b) Un processo di Markov di secondo ordine.

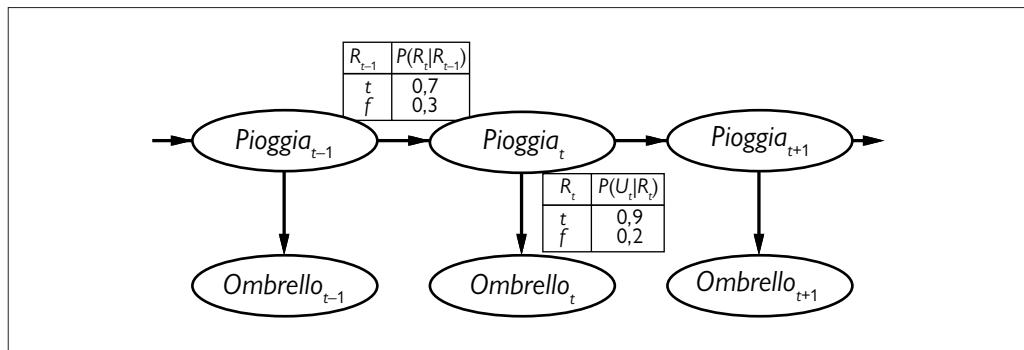


Figura 14.2 Struttura della rete bayesiana e distribuzioni condizionate che descrivono il mondo dell'ombrello. Il modello di transizione è $\mathbf{P}(Pioggia_t|Pioggia_{t-1})$ e quello sensoriale è $\mathbf{P}(Ombrello_t|Pioggia_t)$.

ipotesi di Markov
sensoriale

Passiamo ora al modello sensoriale. Le variabili di evidenza \mathbf{E}_t potrebbero dipendere sia da variabili precedenti sia dalle attuali variabili di stato, ma ogni stato degno di questo nome dovrebbe essere sufficiente per generare i valori sensoriali correnti. Facciamo allora una **ipotesi di Markov sensoriale**:

$$\mathbf{P}(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{1:t-1}) = \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t). \quad (14.2)$$

Quindi, $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$ è il nostro modello sensoriale (talvolta detto modello delle osservazioni). La Figura 14.2 mostra il modello di transizione e il modello sensoriale per l'esempio dell'ombrellino. Notate la direzione della dipendenza tra stati e sensori: le frecce vanno dallo stato attuale del mondo ai valori dei sensori, perché lo stato del mondo *causa* il fatto che i sensori ottengano particolari valori: la pioggia *causa* l'apparizione dell'ombrellino (naturalmente il processo di inferenza va nella direzione opposta: la distinzione tra la direzione delle dipendenze modellate e la direzione dell'inferenza è uno dei principali vantaggi delle reti bayesiane).

Oltre a specificare il modello di transizione e il modello sensoriale, dobbiamo specificare come è iniziato tutto, cioè la distribuzione di probabilità a priori al tempo 0, $\mathbf{P}(\mathbf{X}_0)$. A questo punto disponiamo di una specifica della distribuzione congiunta completa di tutte le variabili (usando l'Equazione (13.2)). Per ogni passo temporale t ,

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \sum_{i=1}^t \mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1}) \mathbf{P}(\mathbf{E}_i | \mathbf{X}_i) \quad (14.3)$$

I tre termini a destra sono il modello di stato iniziale $\mathbf{P}(\mathbf{X}_0)$, il modello di transizione $\mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1})$ e il modello sensoriale $\mathbf{P}(\mathbf{E}_i | \mathbf{X}_i)$. Questa equazione definisce la semantica della famiglia di modelli temporali rappresentati dai tre termini. Notate che le reti bayesiane standard non sono in grado di rappresentare tali modelli, perché richiedono un insieme finito di variabili. La capacità di gestire un insieme infinito di variabili si deve a due fattori: il primo è la definizione dell'insieme infinito usando indici interi; il secondo è l'uso della quantificazione universale implicita (cfr. Paragrafo 8.2) per definire il modello sensoriale e di transizione per ogni passo temporale.

La struttura illustrata nella Figura 14.2 è un processo di Markov del primo ordine – la probabilità di pioggia dipende solo dal fatto che abbia piovuto o meno il giorno prima. Il fatto che questa ipotesi sia o meno ragionevole dipende dal dominio stesso: in un processo di Markov del primo ordine si suppone che le variabili di stato contengano *tutte* le informazioni necessarie per caratterizzare la distribuzione di probabilità dell'istante successivo. Talvolta l'ipotesi è esattamente vera: nel caso in cui una particella stia eseguendo una camminata casuale lungo un'ascissa, modificare la posizione di ± 1 a ogni passo e usare poi la

coordinata x come stato corrente dà luogo a un processo di Markov del primo ordine. Talvolta invece l'ipotesi è vera solo approssimativamente, come nel caso di chi predice la pioggia basandosi solo sulle condizioni atmosferiche del giorno precedente. Esistono due modi per migliorare l'accuratezza dell'approssimazione.

1. Aumentare l'ordine del modello di processo di Markov. Per esempio, potremmo ottenere un modello del secondo ordine aggiungendo $Pioggia_{t-2}$ come genitore di $Pioggia_t$, il che fornirebbe predizioni leggermente più accurate. Per esempio, a Palo Alto in California è molto raro che piova per più di due giorni di seguito.
2. Estendere l'insieme delle variabili di stato. Per esempio, potremmo aggiungere $Stagione_t$ per incorporare archivi storici di stagioni piovose, o $Temperatura_t$, $Umidità_t$ e $Pressione_t$ (magari in un insieme di località) per disporre di un modello fisico delle condizioni atmosferiche che favoriscono la pioggia.

L'Esercizio 14.AUGM vi chiederà di mostrare che la prima soluzione (l'incremento dell'ordine del modello) può sempre essere riformulata come un'estensione dell'insieme delle variabili di stato che mantiene fisso l'ordine. Notate che l'aggiunta di variabili addizionali potrebbe migliorare la potenza predittiva del sistema, ma aumenterà anche i suoi *requisiti*, dato che si dovranno predire valori aggiuntivi. Così, quello che cerchiamo è un insieme di variabili “autosufficiente”, il che in effetti significa che dobbiamo comprendere la “fisica” del processo modellato. La modellazione accurata del processo risulta ovviamente più agevole se possiamo aggiungere nuovi sensori (ovvero misurazioni della temperatura e della pressione) che possano fornirci informazioni dirette sulle nuove variabili di stato.

Considerate per esempio il problema del calcolo della posizione di un robot che si muove casualmente su un piano X–Y. Si potrebbe proporre che posizione e velocità costituiscano un insieme sufficiente di variabili di stato: per calcolare la nuova posizione si applicherebbero semplicemente le leggi di Newton, e la velocità potrebbe cambiare in modo impredicibile. Ma se il robot è alimentato da una batteria, il consumo di quest'ultima tenderebbe ad avere un effetto sistematico sui cambiamenti di velocità. Dato che a sua volta il livello di carica è determinato dalla potenza assorbita in tutte le manovre eseguite, la proprietà di Markov sarebbe violata. Possiamo ripristinarla includendo come variabile di stato in \mathbf{X} , anche il livello di carica $Batteria_t$. Questa soluzione ci permetterebbe di predire il movimento del robot, ma a sua volta richiederebbe un modello di predizione di $Batteria_t$ a partire da $Batteria_{t-1}$ e dalla velocità. In alcuni casi è possibile far ciò in modo affidabile, ma più spesso l'errore si accumula nel tempo. In tal caso, è possibile aumentare l'accuratezza aggiungendo un nuovo sensore per misurare la carica della batteria. Torneremo sull'esempio della batteria nel Paragrafo 14.5.

14.2 Inferenza nei modelli temporali

Avendo definito la struttura di un modello temporale generico, possiamo formulare le attività inferenziali di base che dovremo essere in grado di eseguire.

- **Filtraggio² o stima dello stato:** consiste nel calcolo dello **stato-credenza** $P(\mathbf{X}_t|\mathbf{e}_{1:t})$, ovvero la distribuzione a posteriori sullo stato più recente date tutte le evidenze disponibili. Nell'esempio dell'ombrellino, questo significherebbe calcolare la probabilità che oggi stia piovando date tutte le osservazioni fatte sin qui sull'ombrellino. Il filtraggio è l'attività che un

filtraggio
stima dello stato
stato-credenza

² Il termine “filtraggio” fa riferimento alle origini del problema qui trattato nei primi lavori sull'elaborazione di segnali, in cui il problema era quello di eliminare con un filtro il rumore nel segnale stimando le sue proprietà.

agente razionale svolge per tener traccia dello stato corrente, in modo da poter prendere decisioni razionali. Un calcolo quasi identico fornisce la **verosimiglianza** (*likelihood*) della sequenza delle evidenze $P(\mathbf{e}_{1:t})$.

predizione

- **Predizione:** il calcolo della distribuzione a posteriori dello stato *futuro*, date tutte le evidenze raccolte. Vogliamo calcolare $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ per qualche $k > 0$. Nell'esempio dell'ombrellino, questo potrebbe significare il calcolo della probabilità che possa piovere fra tre giorni, date tutte le osservazioni sull'ombrellino fatte sin qui. La predizione è utile per valutare diversi corsi d'azione possibili in base ai loro esiti attesi.

smoothing

- **Smoothing o regolarizzazione:** è il calcolo della distribuzione a posteriori di uno stato *passato*, date tutte le evidenze raccolte. Vogliamo calcolare $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ per qualche k tale che $0 \leq k < t$. Nell'esempio dell'ombrellino, questo potrebbe significare il calcolo della probabilità che abbia piovuto mercoledì scorso, date tutte le osservazioni fatte sin qui sul portatore dell'ombrellino. Lo smoothing fornisce una stima dello stato al tempo k migliore di quella disponibile a quel momento, perché incorpora una maggiore quantità di evidenze.³
- **Spiegazione più probabile:** data una sequenza di osservazioni, potremmo desiderare di trovare la sequenza di stati che più probabilmente ha generato tali osservazioni. Vogliamo calcolare $\text{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$. Per esempio, se l'ombrellino compare nei primi tre giorni e non nel quarto, la spiegazione più probabile è appunto che abbia piovuto nei primi tre giorni e non nel quarto. Gli algoritmi dedicati a questo compito sono utili in molte applicazioni, incluso il riconoscimento vocale (il cui scopo è trovare la sequenza più probabile di parole, data una serie di suoni) e la ricostruzione di stringhe di bit trasmesse su canali rumorosi.

Oltre a queste attività di inferenza,abbiamo anche:

- **Apprendimento:** il modello di transizione e il modello sensoriale, se non sono ancora noti, possono essere appresi a partire dalle osservazioni. Proprio come nelle reti bayesiane statiche, l'apprendimento nelle reti bayesiane dinamiche può essere svolto come sottoprodotto dell'inferenza: questa infatti fornisce una stima delle transizioni effettivamente avvenute e di quali stati hanno generato le percezioni dei sensori, e tali stime possono essere utilizzate per apprendere il modello. I modelli aggiornati forniscono nuove stime, e il processo itera fino alla convergenza. Il processo di apprendimento può operare attraverso un algoritmo di aggiornamento iterativo denominato EM (*expectation-maximization*, attesa-massimizzazione), oppure può essere generato dall'aggiornamento bayesiano dei parametri del modello date le evidenze. Rimandiamo al Capitolo 20 del Volume 2 per ulteriori dettagli.

Nel seguito di questo paragrafo descriveremo algoritmi generici per le quattro attività di inferenza, indipendenti dal particolare tipo di modello adottato. Nei paragrafi successivi saranno poi descritti alcuni miglioramenti specifici per ogni modello.

14.2.1 Filtraggio e predizione

Come abbiamo sottolineato nel Paragrafo 7.7.3 del Capitolo 7, un algoritmo di filtraggio per essere utile deve mantenere una stima dello stato corrente e aggiornarla, anziché risalire l'intera cronologia delle percezioni a ogni aggiornamento (altrimenti, il costo di ciascun aggiornamento aumenterebbe con il passare del tempo). In altre parole, dato il risultato del

³ In particolare, quando si traccia un oggetto in movimento con osservazioni sulla posizione non accurate, lo smoothing fornisce una traiettoria stimata più liscia rispetto al filtraggio; da qui il suo nome (*smooth* in inglese significa appunto “liscio”).

filtraggio fino al tempo t , l'agente deve calcolare il risultato per $t + 1$ a partire dalla nuova evidenza \mathbf{e}_{t+1} . Perciò abbiamo:

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

per qualche funzione f . Questo processo viene chiamato **stima ricorsiva** (cfr. anche i Paragrafi 4.4 e 7.7.3). Possiamo considerare l'elaborazione suddivisa in due parti: per prima cosa si proietta in avanti la distribuzione dello stato corrente da t a $t + 1$; quindi la si aggiorna in base alla nuova evidenza \mathbf{e}_{t+1} . Questo processo in due fasi emerge in modo piuttosto semplice riordinando la formula:

$$\begin{aligned} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \quad (\text{dividendo le evidenze}) \\ &= \alpha \underbrace{\mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})}_{\substack{\text{aggiornamento} \\ \text{predizione}}} \quad (\text{per la regola di Bayes, dato } \mathbf{e}_{1:t}) \\ &= \alpha \underbrace{\mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})}_{\substack{\text{modello sensoriale}}} \underbrace{\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t)}_{\substack{\text{modello di transizione}}} \underbrace{P(\mathbf{x}_t | \mathbf{e}_{1:t})}_{\substack{\text{ricorsione}}} \quad (\text{per l'ipotesi di Markov sensoriale}). \end{aligned} \quad (14.4)$$

Qui e in tutto il resto del capitolo α indica la costante di normalizzazione utilizzata per far sì che la somma delle probabilità sia sempre 1. Ora inseriamo un'espressione per la predizione a un passo $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})$, ottenuta tramite condizionamento sullo stato corrente \mathbf{X}_t . L'equazione risultante per la stima del nuovo stato è il risultato fondamentale di questo capitolo:

$$\begin{aligned} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \\ &= \alpha \underbrace{\mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})}_{\substack{\text{modello sensoriale}}} \sum_{\mathbf{x}_t} \underbrace{\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t)}_{\substack{\text{modello di transizione}}} \underbrace{P(\mathbf{x}_t | \mathbf{e}_{1:t})}_{\substack{\text{ricorsione}}} \quad (\text{ipotesi di Markov}) \end{aligned} \quad (14.5)$$

In questa espressione, tutti i termini derivano dal modello o dalla stima dello stato precedente. Di conseguenza otteniamo la formulazione ricorsiva desiderata. Possiamo considerare la stima filtrata $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ come un “messaggio” $\mathbf{f}_{1:t}$ propagato in avanti lungo tutta la sequenza, modificato da ogni transizione e aggiornato da ogni nuova osservazione. Il processo è dato da:

$$\mathbf{f}_{1:t+1} = \text{FORWARD}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1}).$$

dove FORWARD implementa l'aggiornamento descritto dall'Equazione (14.5) e il processo inizia con $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0)$. Quando tutte le variabili di stato sono discrete, il tempo richiesto da ogni aggiornamento è costante (cioè indipendente da t), e anche i requisiti spaziali lo sono (le costanti, ovviamente, dipendono dalla dimensione dello spazio degli stati e dal tipo specifico di modello temporale utilizzato). *I requisiti temporali e spaziali per l'aggiornamento devono essere costanti affinché un agente con memoria limitata possa tener traccia della distribuzione dello stato corrente per un tempo indefinito.*



Vediamo ora un paio di passi del processo di filtraggio nel nostro semplice esempio dell'ombrellino (Figura 14.2). Calcoleremo $\mathbf{P}(R_2 | u_{1:2})$ come segue.

- Il giorno 0 non abbiamo osservazioni, solo le credenze a priori della guardia di sicurezza; supponiamo che siano $\mathbf{P}(R_0) = \langle 0,5, 0,5 \rangle$.
- Il giorno 1 appare l'ombrellino, per cui $U_1 = \text{true}$. La predizione da $t = 0$ a $t = 1$ è:

$$\begin{aligned} \mathbf{P}(R_1) &= \sum_{r_0} \mathbf{P}(R_1 | r_0) P(r_0) \\ &= \langle 0,7, 0,3 \rangle \times 0,5 + \langle 0,3, 0,7 \rangle \times 0,5 = \langle 0,5, 0,5 \rangle. \end{aligned}$$

Allora il passo di aggiornamento non fa altro che moltiplicare per la probabilità della evidenza a $t = 1$ e normalizzare, come mostrato nell'Equazione (14.4):

$$\begin{aligned} \mathbf{P}(R_1 | u_1) &= \alpha \mathbf{P}(u_1 | R_1) \mathbf{P}(R_1) = \alpha \langle 0,9, 0,2 \rangle \langle 0,5, 0,5 \rangle \\ &= \alpha \langle 0,45, 0,1 \rangle \approx \langle 0,818, 0,182 \rangle. \end{aligned}$$

- Anche il giorno 2 l'ombrellino compare, per cui $U_2 = \text{true}$. La predizione da $t = 1$ a $t = 2$ è

$$\begin{aligned}\mathbf{P}(R_2 | u_1) &= \sum_{r_1} \mathbf{P}(R_2 | r_1) P(r_1 | u_1) \\ &= \langle 0,7,0,3 \rangle \times 0,818 + \langle 0,3,0,7 \rangle \times 0,182 \approx \langle 0,627, 0,373 \rangle\end{aligned}$$

che, aggiornata con l'evidenza per $t = 2$, ci dà

$$\begin{aligned}\mathbf{P}(R_2 | u_1, u_2) &= \alpha \mathbf{P}(u_2 | R_2) \mathbf{P}(R_2 | u_1) = \alpha \langle 0,9, 0,2 \rangle \langle 0,627, 0,373 \rangle \\ &= \alpha \langle 0,565, 0,075 \rangle \approx \langle 0,883, 0,117 \rangle.\end{aligned}$$

Intuitivamente, la probabilità che piova aumenta dal primo al secondo giorno perché la pioggia persiste. L'Esercizio 14.CONV(a) vi chiederà di investigare ulteriormente questa tendenza.

La **predizione** può essere semplicemente considerata un filtraggio privo dell'aggiunta di nuove evidenze. In effetti, il processo di filtraggio incorpora già una predizione a un passo, da cui è facile derivare la seguente formula ricorsiva per il calcolo dello stato in $t + k + 1$ partendo dalla predizione per $t + k$:

$$\mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \underbrace{\mathbf{P}(\mathbf{X}_{t+k+1} | \mathbf{x}_{t+k})}_{\text{modello di transizione}} \underbrace{\mathbf{P}(\mathbf{x}_{t+k} | \mathbf{e}_{1:t})}_{\text{ricorsione}}. \quad (14.6)$$

Naturalmente il calcolo coinvolge solo il modello di transizione e non quello sensoriale.

tempo di mixing

È interessante considerare cosa succede quando cerchiamo di predire sempre più avanti nel futuro. Come si vede nell'Esercizio 14.CONV(b), la predizione della distribuzione per la pioggia converge al punto fisso $\langle 0,5, 0,5 \rangle$, dopodiché rimane costante per sempre.⁴ Questa è la **distribuzione stazionaria** del processo di Markov definita dal modello di transizione (cfr. anche Paragrafo 13.4.2). Si conoscono molte proprietà di tali distribuzioni e del **tempo di mixing**, che possiamo approssimativamente definire come il tempo necessario per raggiungere il punto fisso. In pratica, è impossibile predire lo stato *reale* per un numero di passi superiore a una piccola frazione del tempo di mixing, a meno che la distribuzione stazionaria stessa abbia un picco in una piccola area dello spazio degli stati. Maggiore è l'incertezza contenuta nel modello di transizione, più breve sarà il tempo di mixing e più ignoto il futuro.

Oltre al filtraggio e alla predizione, possiamo usare una ricorsione in avanti per calcolare la **verosimiglianza** della sequenza di evidenze $P(\mathbf{e}_{1:t})$. Questa si rivela utile per confrontare modelli temporali diversi che potrebbero aver prodotto la stessa sequenza di evidenze (per esempio due diversi modelli per la persistenza della pioggia). Per questa ricorsione usiamo il messaggio di verosimiglianza $\ell_{1:t}(\mathbf{X}_t) = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$. È facile mostrare (Esercizio 14.LIKL) che il calcolo del messaggio è identico a quello per il filtraggio:

$$\ell_{1:t+1} = \text{FORWARD}(\ell_{1:t}, \mathbf{e}_{t+1}).$$

Avendo calcolato $\ell_{1:t}$ possiamo ottenere l'effettiva verosimiglianza eliminando \mathbf{X}_t con una sommatoria:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \ell_{1:t}(\mathbf{x}_t). \quad (14.7)$$

Notate che il messaggio di verosimiglianza rappresenta le probabilità di sequenze di evidenze sempre più lunghe al passare del tempo, che quindi diventano numericamente sempre

⁴ Se si sceglie un giorno arbitrario come $t = 0$, allora ha senso scegliere $\mathbf{P}(\text{Pioggia}_0)$ corrispondente alla distribuzione stazionaria, ecco perché abbiamo scelto $\langle 0,5, 0,5 \rangle$. Se avessimo scelto una probabilità a priori diversa, la distribuzione stazionaria sarebbe comunque stata $\langle 0,5, 0,5 \rangle$.

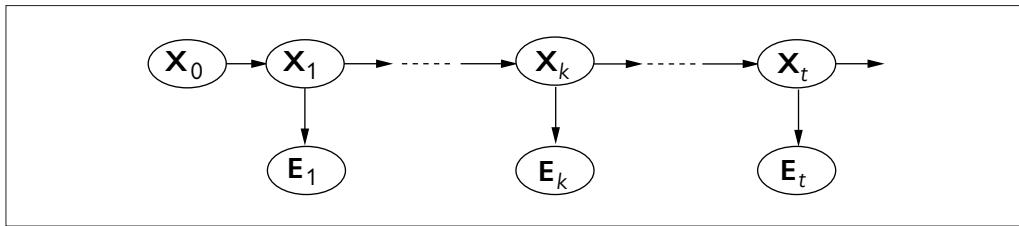


Figura 14.3 Lo smoothing calcola $\mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t})$, la distribuzione a posteriori dello stato in qualche istante passato k data da sequenza completa delle osservazioni da 1 a t .

più piccole, fino a causare problemi di underflow quando si usa l’aritmetica a virgola mobile. Questo è un problema importante nella pratica, ma di cui qui non tratteremo le soluzioni.

14.2.2 Smoothing

Come abbiamo detto in precedenza, lo smoothing (o regolarizzazione) è il processo di calcolo della distribuzione su stati passati, date le evidenze fino al presente; cioè $\mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t})$ per $0 \leq k < t$ (Figura 14.3). Anticipando un altro approccio ricorsivo basato sul passaggio di messaggi, possiamo suddividere il calcolo in due parti: le evidenze fino a k e quelle da $k + 1$ a t ,

$$\begin{aligned}
 \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
 &= \alpha \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \quad (\text{usando la regola di Bayes, dato } \mathbf{e}_{1:k}) \\
 &= \alpha \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) \quad (\text{usando l’indipendenza condizionale}) \\
 &= \alpha \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t},
 \end{aligned} \tag{14.8}$$

dove \times rappresenta la moltiplicazione puntuale tra vettori (componente per componente). Qui abbiamo definito un messaggio “all’indietro” $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k)$, analogo a quello in avanti $\mathbf{f}_{1:k}$. Il messaggio $\mathbf{f}_{1:k}$ può essere calcolato filtrando in avanti da 1 a k , come si vede nell’Equazione (14.5). Il messaggio $\mathbf{b}_{k+1:t}$ può essere calcolato mediante un processo ricorsivo che va *all’indietro* da t :

$$\begin{aligned}
 \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad (\text{condizionando a } \mathbf{X}_{k+1}) \\
 &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad (\text{per l’indipendenza condizionale}) \\
 &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} \underbrace{\mathbf{P}(\mathbf{e}_{k+1} | \mathbf{x}_{k+1})}_{\text{modello sensoriale}} \underbrace{\mathbf{P}(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1})}_{\text{ricorsione}} \underbrace{\mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k)}_{\text{modello di transizione}},
 \end{aligned} \tag{14.9}$$

in cui l’ultimo passo segue dall’indipendenza condizionale di \mathbf{e}_{k+1} ed $\mathbf{e}_{k+2:t}$ dato \mathbf{x}_{k+1} . In questa espressione, tutti i termini derivano dal modello o dal messaggio all’indietro precedente. Abbiamo quindi la formulazione ricorsiva desiderata. In notazione dei messaggi abbiamo:

$$\mathbf{b}_{k+1:t} = \text{BACKWARD}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1}),$$

in cui BACKWARD implementa l’aggiornamento descritto dall’Equazione (14.9). Come nel caso della ricorsione in avanti, il tempo e lo spazio necessari per ogni aggiornamento sono costanti e quindi indipendenti da t .

Possiamo ora vedere che i due termini dell’Equazione (14.8) si possono calcolare entrambi con una ricorsione attraverso il tempo: la prima procede in avanti da 1 a k e usa l’equazione di filtraggio (14.5), la seconda procede all’indietro da t a $k + 1$ e usa l’Equazione (14.9).

Per l'inizializzazione della fase all'indietro abbiamo $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t} | \mathbf{X}_t) = \mathbf{P}(\cdot | \mathbf{X}_t) = \mathbf{1}$, dove $\mathbf{1}$ è un vettore di 1. Il motivo è che $\mathbf{e}_{t+1:t}$ è una sequenza vuota, per cui la probabilità di osservarla è 1.

Applichiamo ora questo algoritmo all'esempio dell'ombrellino, calcolando la stima regolarizzata della probabilità di pioggia al tempo $k = 1$, date le osservazioni dell'ombrellino nei giorni 1 e 2. Dall'Equazione (14.8), risulta:

$$\mathbf{P}(R_1 | u_1, u_2) = \alpha \mathbf{P}(R_1 | u_1) \mathbf{P}(u_2 | R_1). \quad (14.10)$$

Dal processo di filtraggio in avanti che abbiamo descritto, sappiamo già che il primo termine è $\langle 0,818, 0,182 \rangle$. Il secondo termine può essere calcolato applicando la ricorsione all'indietro dell'Equazione (14.9):

$$\begin{aligned} \mathbf{P}(u_2 | R_1) &= \sum_{r_2} P(u_2 | r_2) P(\cdot | r_2) \mathbf{P}(r_2 | R_1) \\ &= (0,9 \times 1 \times \langle 0,7, 0,3 \rangle) + (0,2 \times 1 \times \langle 0,3, 0,7 \rangle) = \langle 0,69, 0,41 \rangle. \end{aligned}$$

Sostituendo nell'Equazione (14.10), troviamo che la stima regolarizzata della pioggia per il giorno 1 è:

$$\mathbf{P}(R_1 | u_1, u_2) = \alpha \langle 0,818, 0,182 \rangle \times \langle 0,69, 0,41 \rangle \approx \langle 0,883, 0,117 \rangle.$$

In questo caso, quindi, la stima regolarizzata per la pioggia al giorno 1 è *più alta* di quella filtrata (0,818). Questo è dovuto al fatto che l'ombrellino rilevato il secondo giorno rende più probabile che in quello stesso giorno stesse piovendo; a sua volta, dato che la pioggia tende a persistere, questo aumenta la probabilità che piovesse anche il primo giorno.

Sia la ricorsione in avanti che quella all'indietro richiedono un tempo costante per ogni passo; di conseguenza la complessità temporale della regolarizzazione rispetto a una evidenza $\mathbf{e}_{1:t}$ è $O(t)$. Questa è la complessità per lo smoothing di un particolare passo temporale k . Se vogliamo regolarizzare l'intera sequenza, un metodo banale è eseguire l'intero processo una volta per ogni passo. Questo risulta in una complessità temporale $O(t^2)$.

Un approccio migliore utilizza un'applicazione semplice della programmazione dinamica per ridurre la complessità a $O(t)$: ne abbiamo avuto un indizio nell'analisi dell'esempio dell'ombrellino qui sopra, in cui siamo stati in grado di riusare i risultati della fase di filtraggio in avanti. La chiave dell'algoritmo con tempo lineare sta nel *memorizzare i risultati* del filtraggio in avanti su tutta la sequenza. Fatto questo possiamo eseguire la ricorsione all'indietro da t fino a 1, calcolando la stima regolarizzata di ogni passo k dal messaggio all'indietro calcolato $\mathbf{b}_{k+1:t}$ e quello in avanti memorizzato $\mathbf{f}_{1:k}$. L'algoritmo, appropriatamente denominato **algoritmo forward-backward**, è mostrato nella Figura 14.4.

algoritmo forward-backward

Il lettore attento avrà notato che la struttura della rete bayesiana della Figura 14.3 è un *polialbero* (cfr. Paragrafo 13.3.3 per la definizione). Questo significa che un'applicazione diretta dell'algoritmo di clustering ha anch'essa come risultato un algoritmo che, in tempo lineare, calcola stime regolarizzate dell'intera sequenza. Oggi si è compreso che l'algoritmo forward-backward è in effetti un caso speciale della propagazione di polialberi utilizzata nei metodi di clustering (benché le due tecniche siano state sviluppate indipendentemente).

L'algoritmo forward-backward costituisce il fondamento computazionale di molte applicazioni che devono gestire sequenze di osservazioni rumorose. Così come lo abbiamo descritto finora, presenta due limitazioni pratiche: la prima è che la sua complessità spaziale può essere troppo alta quando lo spazio degli stati è grande e le sequenze lunghe. Lo spazio richiesto è $O(|\mathbf{f}|t)$, dove $|\mathbf{f}|$ sono le dimensioni della rappresentazione del messaggio in avanti. I requisiti spaziali possono essere ridotti a $O(|\mathbf{f}| \log t)$ al costo di un incremento della complessità temporale di un fattore $\log t$, come dimostra l'Esercizio 14.ISLE. In alcuni casi (cfr. Paragrafo 14.3), è possibile utilizzare un algoritmo con requisiti spaziali costanti.

function FORWARD-BACKWARD(**ev**, *priori*) **returns** un vettore di distribuzioni di probabilità

inputs: **ev**, un vettore di valori di evidenza per i passi temporali $1, \dots, t$

priori, la distribuzione a priori dello stato iniziale $\mathbf{P}(\mathbf{X}_0)$

local variables: **fv**, un vettore di messaggi in avanti per i passi $0, \dots, t$

b, una rappresentazione del messaggio all'indietro,

inizialmente contenente tutti 1

sv, un vettore di stime regolarizzate dei passi $1, \dots, t$

```

fv[0]  $\leftarrow$  priori
for  $i = 1$  to  $t$  do
    fv[ $i$ ]  $\leftarrow$  FORWARD (fv[ $i - 1$ ], ev[ $i$ ])
for  $i = t$  down to 1 do
    sv[ $i$ ]  $\leftarrow$  NORMALIZZA(fv[ $i$ ]  $\times$  b)
    b  $\leftarrow$  BACKWARD (b, ev[ $i$ ])
return sv

```

Figura 14.4 L'algoritmo forward–backward per lo smoothing: calcola le probabilità a posteriori di una sequenza di stati data una sequenza di osservazioni. Gli operatori FORWARD e BACKWARD sono definiti rispettivamente dalle Equazioni (14.5) e (14.9).

La seconda limitazione dell'algoritmo base è la sua incapacità di lavorare in un ambiente *online* in cui si devono calcolare continuamente stime regolarizzate di istanti passati man mano che nuove osservazioni si aggiungono alla fine della sequenza. Il requisito più comune è quello dello **smoothing (o regolarizzazione) a ritardo fisso** (*fixed-lag smoothing*), che richiede il calcolo della stima regolarizzata $\mathbf{P}(\mathbf{X}_{t-d}|\mathbf{e}_{1:t})$ per un d prefissato. In altre parole, si esegue lo smoothing della time slice che si trova d istanti prima del tempo corrente t ; all'avanzare di t , la regolarizzazione deve tenere il passo. Ovviamente è possibile eseguire l'algoritmo forward–backward su una “finestra” larga d passi ogni volta che viene aggiunta una nuova osservazione, ma questo sembra inefficiente. Nel Paragrafo 14.3 vedremo che, in certi casi, la regolarizzazione a ritardo fisso può essere effettuata in un tempo costante per ogni aggiornamento, indipendente dal ritardo d .

smoothing a ritardo fisso

14.2.3 Trovare la sequenza più probabile

Supponiamo che *[true, true, false, true, true]* sia la sequenza di apparizioni dell'ombrellino osservata nei primi cinque giorni di lavoro della guardia di sicurezza. Qual è la sequenza di condizioni atmosferiche più probabile per giustificare tali osservazioni? L'assenza dell'ombrellino nel terzo giorno significa che non stava piovendo, o forse il direttore si è dimenticato di portarlo? E se non pioveva il giorno 3, forse (dato che il tempo tende a persistere) non pioveva neppure il giorno 4, ma il direttore ha portato l'ombrellino per ogni evenienza. In tutto ci sono 2^5 sequenze possibili. Esiste un modo per trovare la più probabile senza enumerarle tutte e calcolarne la verosimiglianza?

Potremmo provare la seguente procedura in tempo lineare: usare lo smoothing per trovare la distribuzione a posteriori delle condizioni atmosferiche in ogni passo temporale; fatto questo possiamo costruire la sequenza, scegliendo per ogni passo le condizioni più probabili secondo la distribuzione. Un approccio simile dovrebbe far suonare un campanello d'allarme nella testa dei lettori, perché le distribuzioni a posteriori calcolate mediante la regolarizzazione si riferiscono a passi *singoli*, laddove per trovare la *sequenza* più probabile dobbiamo

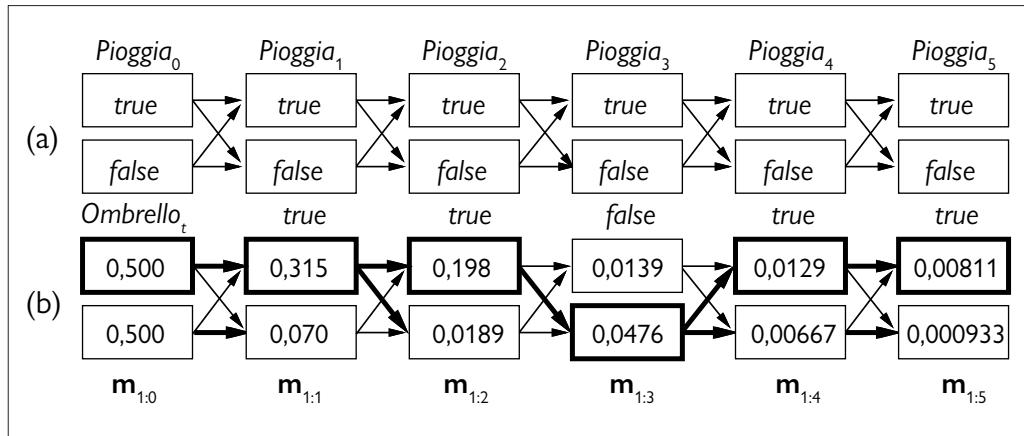


Figura 14.5 (a) Le possibili sequenze di stati per $Pioggia_t$ possono essere viste come cammini in un grafo di stati possibili a ogni passo temporale (gli stati sono indicati da rettangoli per evitare confusioni con le reti bayesiane). (b) Il funzionamento dell'algoritmo di Viterbi per la sequenza di osservazioni dell'ombrello [true, true, false, true, true], dove l'evidenza inizia al tempo 1. Per ogni t abbiamo riportato i valori del messaggio $\mathbf{m}_{1:t}$, che fornisce la probabilità della migliore sequenza che raggiunge ogni stato al tempo t . Inoltre, per ogni stato la freccia più spessa che conduce a esso indica il suo miglior predecessore, misurato in base al prodotto della probabilità della precedente sequenza per la probabilità di transizione. Per ottenere la sequenza più probabile basta seguire all'indietro le frecce in grassetto, partendo dallo stato più probabile in $\mathbf{m}_{1:5}$.

considerare le probabilità *congiunte* su tutti i passi temporali. I risultati in effetti possono essere ben diversi (cfr. Esercizio 14.VITE).

In effetti *esiste* un algoritmo in tempo lineare per trovare la sequenza più probabile, ma richiede una riflessione più articolata. L'algoritmo si basa sulla stessa proprietà di Markov che ci ha permesso di ottenere algoritmi efficienti per il filtraggio e la regolarizzazione. L'idea è considerare ogni sequenza come un *cammino* attraverso un grafo i cui nodi rappresentano i possibili *stati* in ogni passo temporale. La Figura 14.5(a) mostra un grafo siffatto per il mondo dell'ombrello. Ora considerate il compito di trovare il cammino più probabile attraverso questo grafo, dove la probabilità di ogni cammino è il prodotto delle probabilità di transizione lungo il cammino stesso per le probabilità delle osservazioni rilevate in ogni stato. Concentriamoci in particolare sui cammini che raggiungono lo stato $Pioggia_5 = \text{true}$. Grazie alla proprietà di Markov, ne consegue che il cammino più probabile per lo stato $Pioggia_5 = \text{true}$ consiste nel cammino più probabile verso *un qualche* stato all'istante 4 seguito da una transizione da lì a $Pioggia_5 = \text{true}$; e lo stato all'istante 4 che diventerà parte del cammino per $Pioggia_5 = \text{true}$ è quello che massimizza la probabilità di tale cammino. In altre parole, *c'è una relazione ricorsiva tra i cammini più probabili verso ogni stato \mathbf{x}_{t+1} e i cammini più probabili verso ogni stato \mathbf{x}_t* .

Possiamo usare direttamente questa proprietà per costruire un algoritmo ricorsivo che calcoli il cammino più probabile data l'evidenza. Utilizzeremo un messaggio calcolato ricorsivamente, $m_{1:t}$, come il messaggio in avanti $\mathbf{f}_{1:t}$ nell'algoritmo del filtraggio. Il messaggio è definito come segue:⁵

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_{1:t-1}} \mathbf{P}(\mathbf{x}_{1:t-1}, \mathbf{X}_t, \mathbf{e}_{1:t}).$$

⁵ Notate che queste non sono le probabilità dei cammini più probabili per raggiungere gli stati \mathbf{X}_t data l'evidenza, che sarebbero le probabilità condizionate $\max_{\mathbf{x}_{1:t-1}} \mathbf{P}(\mathbf{x}_{1:t-1}, \mathbf{X}_t | \mathbf{e}_{1:t})$; ma i due vettori sono correlati per un fattore costante $P(\mathbf{e}_{1:t})$. La differenza è trascurabile perché l'operatore max non tiene conto di fattori costanti. Otteniamo una ricorsione leggermente più semplice con $\mathbf{m}_{1:t}$ definito in questo modo.

Per ottenere la relazione ricorsiva tra $\mathbf{m}_{1:t+1}$ e $\mathbf{m}_{1:t}$ possiamo ripetere più o meno gli stessi passaggi usati per l'Equazione (14.5):

$$\begin{aligned}\mathbf{m}_{1:t+1} &= \max_{\mathbf{x}_{1:t}} \mathbf{P}(\mathbf{x}_{1:t}, \mathbf{X}_{t+1}, \mathbf{e}_{1:t+1}) = \max_{\mathbf{x}_{1:t}} \mathbf{P}(\mathbf{x}_{1:t}, \mathbf{X}_{t+1}, \mathbf{e}_{1:t}, e_{t+1}) \\ &= \max_{\mathbf{x}_{1:t}} \mathbf{P}(e_{t+1} | \mathbf{x}_{1:t}, \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{x}_{1:t}, \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \\ &= \mathbf{P}(e_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_{1:t}} \mathbf{P}(\mathbf{X}_{t+1}, |\mathbf{x}_t) P(\mathbf{x}_{1:t}, \mathbf{e}_{1:t}) \\ &= \mathbf{P}(e_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}, |\mathbf{x}_t) \max_{\mathbf{x}_{1:t-1}} \mathbf{P}(\mathbf{x}_{1:t-1}, \mathbf{x}_t, \mathbf{e}_{1:t})\end{aligned}\quad (14.11)$$

dove il termine finale $\max_{\mathbf{x}_{1:t-1}} P(\mathbf{x}_{1:t-1}, \mathbf{x}_t, \mathbf{e}_{1:t})$ è esattamente l'elemento corrispondente allo stato \mathbf{x}_t nel vettore messaggio $\mathbf{m}_{1:t}$. L'Equazione (14.11) è sostanzialmente identica all'equazione di filtraggio (14.5) a parte il fatto che la somma su \mathbf{x}_t dell'Equazione (14.5) diventa la massimizzazione su \mathbf{x}_t nell'Equazione (14.11), e non c'è la costante di normalizzazione α nell'Equazione (14.11). Quindi, l'algoritmo per calcolare la sequenza più probabile è simile all'algoritmo di filtraggio: inizia al tempo 0 con $\mathbf{m}_{1:0} = \mathbf{P}(\mathbf{X}_0)$ e poi procede in avanti lungo la sequenza, calcolando il messaggio \mathbf{m} a ogni passo mediante l'Equazione (14.11). La Figura 14.5(b) illustra come avanza il calcolo.

Alla fine della sequenza di osservazioni, $\mathbf{m}_{1:t}$ conterrà la probabilità della sequenza più probabile che raggiunge *ognuno* degli stati finali. Risulta così facile selezionare lo stato finale della sequenza più probabile in assoluto (lo stato evidenziato in grassetto al passo 5). Per trovare l'effettiva sequenza, invece di limitarsi a calcolare la sua probabilità, l'algoritmo dovrà anche registrare, per ogni stato, lo stato migliore che conduce a esso – le frecce più spesse nella Figura 14.5(b). La sequenza ottima è identificata seguendo queste frecce più spesse all'indietro dal migliore stato finale.

L'algoritmo che abbiamo appena descritto è chiamato **algoritmo di Viterbi** dal nome del suo inventore, Andrea Viterbi. Come quello del filtraggio la sua complessità temporale è lineare in t , la lunghezza della sequenza. A differenza del filtraggio, che opera in spazio costante, anche i requisiti spaziali sono lineari in t . Questo è dovuto al fatto che l'algoritmo di Viterbi deve tenere in memoria i puntatori che identificano la sequenza migliore che conduce a ogni stato.

algoritmo di Viterbi

Un ultimo aspetto di carattere pratico: per l'algoritmo di Viterbi l'underflow numerico costituisce un problema rilevante. Nella Figura 14.5(b) le probabilità diventano sempre più piccole, e questo è solo un esempio molto semplice. Le applicazioni reali per l'analisi del DNA o la decodifica di messaggi potrebbero avere migliaia o milioni di passi. Una possibile soluzione è semplicemente quella di normalizzare \mathbf{m} a ogni passo; questo non ha effetto sulla correttezza perché $\max(cx, cy) = c \cdot \max(x, y)$. Un'altra soluzione consiste nell'usare probabilità logaritmiche ovunque per trasformare le moltiplicazioni in addizioni. Anche in questo caso la correttezza rimane invariata perché la funzione logaritmo è monotona, perciò $\max(\log x, \log y) = \log \max(x, y)$.

14.3 Modelli di Markov nascosti

Nel paragrafo precedente abbiamo sviluppato algoritmi per il ragionamento temporale probabilistico all'interno di un'infrastruttura generale, indipendente da una forma specifica del modello di transizione e di quello sensoriale e anche dalla natura delle variabili di stato e di evidenza. In questo paragrafo, e nei due successivi, discuteremo modelli e applicazioni più concrete per illustrare la potenza degli algoritmi base e, in alcuni casi, permettere ulteriori sviluppi.

Cominceremo con il **modello di Markov nascosto** o **HMM** (*hidden Markov model*). Un HMM è un modello temporale probabilistico in cui lo stato del processo è descritto da una *singola* variabile casuale *discreta*, i cui possibili valori corrispondono agli stati possibili del

modello di Markov nascosto

mondo. Il nostro esempio del mondo dell’ombrellino è quindi un HMM, dato che ha una sola variabile di stato: *Pioggia_t*. E se si ha un modello con due o più variabili di stato? Lo si può comunque fare rientrare nell’architettura HMM combinando le variabili in una singola “megavariabile” i cui valori sono costituiti da tutte le possibili tuple di valori delle variabili di stato individuali. Come vedremo, la natura limitata dei modelli HMM permette un’implementazione molto semplice ed elegante di tutti gli algoritmi base.⁶

I modelli HMM richiedono che lo *stato* sia una singola variabile discreta, ma non esiste un vincolo corrispondente sulle variabili *di evidenza*, dato che queste sono sempre osservate, quindi non vi è necessità di mantenere traccia di una distribuzione sui loro valori (se una variabile non è osservata, può semplicemente essere rimossa dal modello per il passo temporale di interesse). Possono esserci molte variabili di evidenza, discrete e continue.

14.3.1 Algoritmi semplificati basati su matrici

Con una singola variabile di stato discreta X_t possiamo dare una forma concreta alle rappresentazioni del modello di transizione, del modello sensoriale e dei messaggi in avanti e all’indietro. Supponiamo di denotare i possibili stati della variabile X_t con i valori interi 1, …, S , dove S è il numero di stati possibili. Il modello di transizione $\mathbf{P}(X_t|X_{t-1})$ diventa una matrice \mathbf{T} di dimensioni $S \times S$, in cui:

$$\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i).$$

ovvero, \mathbf{T}_{ij} è la probabilità di una transizione dallo stato i allo stato j . Per esempio, se numeriamo gli stati *Pioggia = true* e *Pioggia = false* come 1 e 2, rispettivamente, allora la matrice di transizione per il mondo dell’ombrellino, definita nella Figura 14.2, è:

$$\mathbf{T} = \mathbf{P}(X_t | X_{t-1}) = \begin{pmatrix} 0,7 & 0,3 \\ 0,3 & 0,7 \end{pmatrix}.$$

Anche il modello sensoriale è espresso in forma di matrice. In questo caso, dato che il valore della variabile di evidenza E_t è noto al tempo t (lo chiamiamo e_t), dobbiamo soltanto specificare, per ogni stato, la probabilità che tale stato causi l’apparire di e_t . Ci serve $P(e_t | X_t = i)$ per ogni stato i . Per comodità di calcolo inseriamo questi valori in una **matrice di osservazione** diagonale $S \times S$, \mathbf{O}_t , una per ogni passo temporale. L’elemento i -esimo sulla diagonale di \mathbf{O}_t è $P(e_t | X_t = i)$ e tutti gli altri elementi sono 0. Per esempio, al giorno 1 nel mondo dell’ombrellino della Figura 14.5 abbiamo $U_1 = \text{true}$, mentre al giorno 3 abbiamo $U_1 = \text{false}$. Abbiamo quindi:

$$\mathbf{O}_1 = \begin{pmatrix} 0,9 & 0 \\ 0 & 0,2 \end{pmatrix}; \quad \mathbf{O}_3 = \begin{pmatrix} 0,1 & 0 \\ 0 & 0,8 \end{pmatrix}.$$

Ora, se usiamo i vettori colonna per rappresentare i messaggi in avanti e all’indietro, tutti i calcoli si riducono a semplici operazioni tra matrici e vettori. L’equazione in avanti (14.5) diventa:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \tag{14.12}$$

e quella all’indietro (14.9) diventa:

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}. \tag{14.13}$$

matrice di osservazione

⁶ Chi ha poca familiarità con le operazioni di base sui vettori e sulle matrici può consultare l’Appendice A prima di procedere nella lettura.

Da queste equazioni possiamo vedere che la complessità temporale dell'algoritmo forward-backward (Figura 14.4) applicata a una sequenza di lunghezza t è $O(S^2t)$, dato che ogni passo richiede di moltiplicare un vettore di S elementi per una matrice $S \times S$. I requisiti spaziali sono $O(St)$, poiché il passaggio in avanti memorizza t vettori di lunghezza S .

Oltre a fornire una descrizione elegante degli algoritmi di filtraggio e smoothing per gli HMM, la formulazione matriciale rivela alcune possibili migliorie. La prima è una semplice variazione dell'algoritmo forward–backward che permette di eseguire lo smoothing con un'occupazione spaziale *costante*, indipendentemente dalla lunghezza della sequenza. L'idea è che la regolarizzazione per una qualsiasi time slice k richiede, in base all'Equazione (14.8), la presenza simultanea sia del messaggio in avanti $\mathbf{f}_{1:k}$ che di quello all'indietro $\mathbf{b}_{k+1:t}$. L'algoritmo forward–backward raggiunge lo scopo memorizzando gli \mathbf{f} calcolati nel passaggio in avanti, in modo che siano disponibili durante il successivo passaggio all'indietro. Per ottenere lo stesso risultato con una sola passata si può propagare sia \mathbf{f} che \mathbf{b} nella stessa direzione. Per esempio, il messaggio “in avanti” \mathbf{f} può essere passato all'indietro se modifichiamo l'Equazione (14.12) in modo che lavori nell'altra direzione:

$$\mathbf{f}_{1:t} = \alpha' (\mathbf{T}^\top)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1}.$$

L'algoritmo di smoothing modificato esegue per prima cosa un passo standard in avanti per calcolare $\mathbf{f}_{1:t}$ (dimenticando tutti i risultati intermedi) e poi effettua una passata all'indietro per \mathbf{b} e \mathbf{f} insieme, utilizzandoli entrambi per calcolare la stima regolarizzata a ogni passo. Dato che è necessaria una sola copia di ogni messaggio, i requisiti di memoria sono costanti (cioè indipendenti dalla lunghezza della sequenza t). Quest'algoritmo ha due significative limitazioni: richiede che la matrice di transizione sia invertibile e che il modello sensoriale non abbia zeri (il che significa che in ogni stato dev'essere possibile eseguire tutte le osservazioni).

Una seconda area in cui la formulazione matriciale rivela un miglioramento è lo smoothing *online* a ritardo fisso. Il fatto che la regolarizzazione possa essere calcolata in uno spazio costante suggerisce che debba esistere un algoritmo ricorsivo efficiente per effettuarla online; un algoritmo, cioè, la cui complessità temporale sia indipendente dalla lunghezza del ritardo. Supponiamo che il ritardo sia pari a d ; in altre parole, al tempo t vogliamo regolarizzare all'istante $t-d$. Per l'Equazione (14.8), si deve calcolare

$$\alpha \mathbf{f}_{1:t-d} \mathbf{b}_{t-d+1:t}$$

per l'istante $t-d$. Quando arriva una nuova osservazione, dobbiamo calcolare

$$\alpha \mathbf{f}_{1:t-d+1} \mathbf{b}_{t-d+2:t+1}$$

per l'istante $t-d+1$. Com'è possibile farlo in modo incrementale? Per prima cosa possiamo calcolare $\mathbf{f}_{1:t-d+1}$ da $\mathbf{f}_{1:t-d}$, usando il processo standard di filtraggio: l'Equazione (14.5).

Calcolare incrementalmente il messaggio all'indietro è più difficile, perché non esiste una relazione semplice tra il vecchio messaggio $\mathbf{b}_{t-d+1:t}$ e quello nuovo $\mathbf{b}_{t-d+2:t+1}$. Esamineremo invece la relazione tra il vecchio messaggio all'indietro $\mathbf{b}_{t-d+1:t}$ e quello all'inizio della sequenza, $\mathbf{b}_{t+1:t}$. Per far questo applichiamo l'Equazione (14.13) d volte per ottenere

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^t \mathbf{T} \mathbf{O}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t} \mathbf{1}, \quad (14.14)$$

dove la matrice $\mathbf{B}_{t-d+1:t}$ è il prodotto della sequenza di matrici \mathbf{T} e \mathbf{O} e $\mathbf{1}$ è un vettore di 1. \mathbf{B} può essere considerata come un “operatore di trasformazione” che prende un messaggio all'indietro e restituisce un messaggio precedente. Un'equazione simile vale per il nuovo messaggio all'indietro *dopo* che è arrivata l'osservazione successiva:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=t-d+2}^{t+1} \mathbf{T} \mathbf{O}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1} \mathbf{1}, \quad (14.15)$$

function SMOOTHING-RITARDO-FISSO(e_t, hmm, d) **returns** una distribuzione su \mathbf{X}_{t-d}

inputs: e_t , l'evidenza corrente all'istante t
 hmm , un modello di Markov nascosto
con matrice di transizione \mathbf{T} di dimensioni $S \times S$
 d , la lunghezza del ritardo di regolarizzazione

static: t , l'istante corrente, inizialmente 1
 \mathbf{f} , il messaggio in avanti $\mathbf{P}(X_t|e_{1:t})$, inizialmente $hmm.PRIORI$
 \mathbf{B} , la matrice di trasformazione all'indietro a d passi,
inizialmente una matrice identità
 $e_{t-d:t}$, lista a doppio ingresso delle evidenze da $t - d$ a t , inizialmente vuota

local variables: \mathbf{O}_{t-d} \mathbf{O}_t , matrici diagonali contenenti informazioni relative al modello sensoriale

inserisci e_t alla fine di $e_{t-d:t}$
 $\mathbf{O}_t \leftarrow$ la matrice diagonale che contiene $\mathbf{P}(e_t|X_t)$
if $t > d$ **then**
 $\mathbf{f} \leftarrow$ FORWARD(\mathbf{f}, e_{t-d})
 rimuovi e_{t-d-1} dalla testa di $e_{t-d:t}$
 $\mathbf{O}_{t-d} \leftarrow$ la matrice diagonale che contiene $\mathbf{P}(e_{t-d}|X_{t-d})$
 $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1} \mathbf{T}^{-1} \mathbf{B} \mathbf{T} \mathbf{O}_t$
else $\mathbf{B} \leftarrow \mathbf{B} \mathbf{T} \mathbf{O}_t$
 $t \leftarrow t + 1$
if $t > d$ **then return** NORMALIZZA($\mathbf{f} \times \mathbf{B} \mathbf{1}$) **else return** null

Figura 14.6 Un algoritmo per lo smoothing con un ritardo fisso di d passi, implementato in una modalità online che fornisce in uscita la nuova stima regolarizzata date le osservazioni nel nuovo passo temporale. Notate che l'output finale NORMALIZZA($\mathbf{f} \times \mathbf{B} \mathbf{1}$) è semplicemente $\alpha \mathbf{f} \times \mathbf{b}$, per l'Equazione (14.14).

Esaminando le produttorie nelle Equazioni (14.14) e (14.15), vediamo che hanno una relazione semplice: per ottenere il secondo prodotto si “divide” il primo prodotto per il primo elemento $\mathbf{T} \mathbf{O}_{t-d+1}$ e lo si moltiplica per il nuovo ultimo elemento $\mathbf{T} \mathbf{O}_{t+1}$. Nel linguaggio delle matrici, quindi, c'è una relazione semplice tra la matrice \mathbf{B} vecchia e quella nuova:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1} \mathbf{T}^{-1} \mathbf{B}_{t-d+1:t} \mathbf{T} \mathbf{O}_{t+1}. \quad (14.16)$$

Questa equazione fornisce un aggiornamento incrementale per la matrice \mathbf{B} , che a sua volta (attraverso l'Equazione (14.15)) ci permette di calcolare il nuovo messaggio all'indietro $\mathbf{b}_{t-d+2:t+1}$. L'algoritmo completo, che richiede la memorizzazione e l'aggiornamento di \mathbf{f} e \mathbf{B} , è mostrato nella Figura 14.6.

14.3.2 Un esempio di modello di Markov nascosto: localizzazione

Nel Paragrafo 4.4.4 del Capitolo 4 abbiamo introdotto una forma semplice del problema di **localizzazione** per il mondo dell'aspirapolvere. In quella versione il robot aveva azioni non deterministiche e i suoi sensori erano in grado di rilevare perfettamente se vi erano ostacoli posti immediatamente a nord, sud, est e ovest. Lo stato-credenza del robot era l'insieme delle posizioni in cui poteva trovarsi.

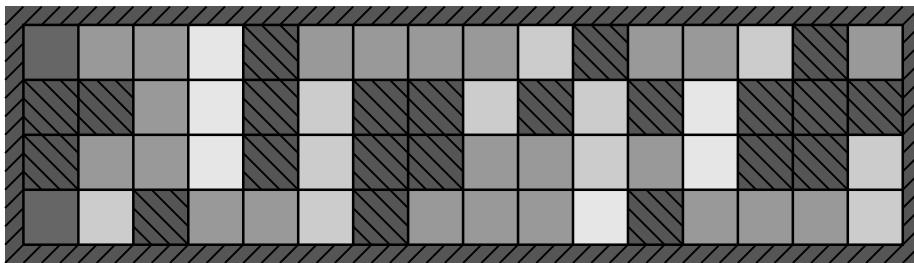
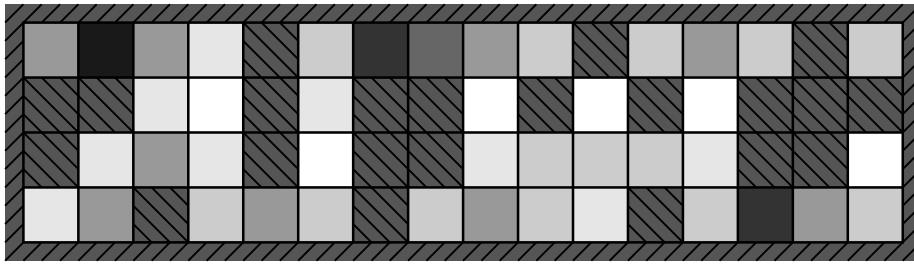
(a) Distribuzione a posteriori sulla posizione del robot dopo $E_1 = 1011$.(b) Distribuzione a posteriori sulla posizione del robot dopo $E_1 = 1011, E_2 = 1010$.

Figura 14.7 Distribuzione a posteriori sulla posizione del robot: (a) dopo un'osservazione $E_1 = 1011$ (cioè con ostacoli a nord, sud e ovest); (b) dopo uno spostamento casuale in una posizione adiacente e una seconda osservazione $E_2 = 1010$ (cioè con ostacoli a nord e sud). Il livello di grigio di ciascuna casella corrisponde alla probabilità che il robot si trovi in quella posizione: a grigio più scuro corrisponde una maggiore probabilità. Il tasso di errore del sensore per ogni bit è $\varepsilon = 0,2$.

Ora rendiamo un po' più realistico il problema introducendo la possibilità di errori nei sensori e formalizzando il concetto che il robot si muove a caso: la probabilità che si sposti in uno qualsiasi dei riquadri adiacenti è la stessa. La variabile di stato X_t rappresenta la posizione del robot sulla griglia discreta; il suo dominio è l'insieme dei riquadri vuoti, che etichettiamo con i numeri interi $\{1, \dots, S\}$. Sia $VICINI(i)$ l'insieme dei riquadri vuoti adiacenti a i e $N(i)$ il numero di elementi di tale insieme. Allora il modello di transizione per l'azione *Muovi* afferma che il robot può andare con pari probabilità in uno qualsiasi dei riquadri vicini:

$$P(X_{t+1} = j | X_t = i) = \mathbf{T}_{ij} = \begin{cases} 1/N(i) & \text{se } j \in VICINI(i) \\ 0 & \text{altrimenti.} \end{cases}$$

Non sappiamo da dove parte il robot, perciò ipotizzeremo una distribuzione uniforme su tutti i riquadri: $P(X_0 = i) = 1/S$. Per il particolare ambiente che consideriamo (Figura 14.7), $S = 42$ e la matrice di transizione \mathbf{T} ha $42 \times 42 = 1764$ elementi.

La variabile sensore E_t ha 16 possibili valori, ognuno dei quali è una sequenza di quattro bit che indica la presenza o l'assenza di un ostacolo in ognuna delle direzioni corrispondenti nord, est, sud, ovest. Per esempio 1010 significa che i sensori a nord e a sud indicano la presenza di un ostacolo e quelli a est e a ovest no. Supponiamo che il tasso di errore di ogni sensore sia ε e che gli errori si verifichino in modo indipendente per le quattro direzioni dei sensori. In tal caso, la probabilità di ottenere quattro bit tutti giusti è $(1 - \varepsilon)^4$ e la probabilità

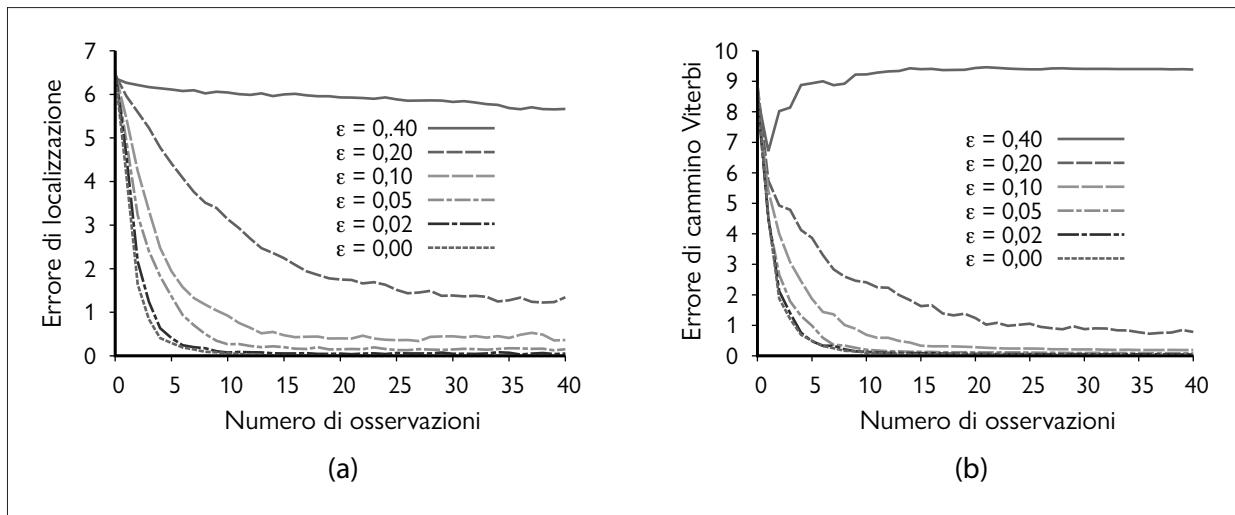


Figura 14.8 Prestazione della localizzazione HMM in funzione della lunghezza della sequenza di osservazione per diversi valori della probabilità di errore del sensore ϵ ; media su 400 esecuzioni. (a) L'errore di localizzazione, definito come distanza Manhattan dalla vera posizione. (b) L'errore di cammino Viterbi, definito come distanza Manhattan media degli stati sul cammino Viterbi dagli stati corrispondenti sul cammino vero.

che siano tutti sbagliati è ϵ^4 . Inoltre, se d_{it} è la discrepanza (il numero di bit diversi) tra i valori veri per il riquadro i e la lettura corrente e_t , allora la probabilità che un robot nel riquadro i riceva una lettura del sensore e_t è:

$$P(E_t = e_t | X_t = i) = (\mathbf{O}_t)_{ii} = (1-\epsilon)^{4-d_{it}}\epsilon^{d_{it}}.$$

Per esempio, la probabilità che un riquadro con ostacoli a nord e a sud produca una lettura di 1110 è $(1-\epsilon)^3\epsilon^1$.

Date le matrici \mathbf{T} e \mathbf{O}_t , il robot può usare l'Equazione (14.12) per calcolare la distribuzione a posteriori sulle posizioni, cioè per determinare dove si trova. La Figura 14.7 mostra le distribuzioni $\mathbf{P}(X_1 | E_1 = 1011)$ e $\mathbf{P}(X_2 | E_1 = 1011, E_2 = 1010)$. È la stessa situazione che abbiamo visto precedentemente nella Figura 4.18 del Capitolo 4, ma in quel caso abbiamo usato il filtraggio logico per trovare le posizioni *possibili*, ipotizzando una capacità di rilevamento dei sensori perfetta. Quelle stesse posizioni sono ancora le più *probabili* adesso che utilizziamo sensori con disturbo, ma ora *ogni* posizione ha una probabilità non nulla, perché ogni posizione può produrre qualsiasi valore dei sensori.

Oltre a ricorrere al filtraggio per stimare la sua posizione corrente, il robot può usare lo smoothing (Equazione (14.13)) per determinare dove si trovava in qualsiasi istante temporale passato – per esempio quando è partito al tempo 0 – e può servirsi dell'algoritmo di Viterbi per determinare il cammino più probabile che lo ha portato dove si trova ora. La Figura 14.8 mostra l'errore di localizzazione e l'errore del cammino di Viterbi per vari valori del tasso di errore del sensore ϵ . Anche quando ϵ è 0,20 – il che significa che la lettura complessiva dei sensori è sbagliata per il 59% del tempo – il robot è solitamente in grado di determinare la sua posizione entro due riquadri dopo 20 osservazioni. Questo si deve alla capacità dell'algoritmo di integrare evidenze nel tempo e di tenere conto dei vincoli probabilistici imposti dal modello di transizione sulla sequenza di posizioni. Quando ϵ è minore o uguale a 0,10, al robot bastano poche osservazioni per determinare dove si trova e tracciare accuratamente la sua posizione. Quando ϵ è 0,40, sia l'errore di localizzazione sia quello del cammino di Viterbi rimangono grandi; in altre parole, il robot si è perso. Il motivo è che un sen-

sore con una probabilità di errore di 0,40 fornisce troppo poche informazioni per controbilanciare la perdita di informazioni sulla posizione del robot dovuta al fatto che si muove in modo casuale e impredicibile.

La variabile di stato per l'esempio considerato in questo paragrafo indica una posizione fisica nel mondo. Altri problemi potrebbero naturalmente comprendere altri aspetti del mondo. L'Esercizio 14.ROOM vi chiede di considerare una versione del robot aspirapolvere che si muove in direzione rettilinea finché può e cambia direzione soltanto quando incontra un ostacolo. In un modello per questo robot, ogni stato è costituito da una coppia (*posizione, direzione*). Per l'ambiente della Figura 14.7, in cui ci sono 42 riquadri vuoti, si hanno in totale 168 stati e una matrice di transizione di $168^2 = 28.224$ elementi, un numero ancora gestibile.

Se aggiungiamo la possibilità che vi sia dello sporco in ognuno dei 42 riquadri, il numero di stati viene moltiplicato per 2^{42} e la matrice di transizione ha oltre 10^{29} elementi, un numero non più gestibile. In generale, se lo stato è composto da n variabili discrete con al più d valori ciascuna, la matrice di transizione HMM corrispondente avrà dimensione $O(d^{2n})$ e il tempo di calcolo per un aggiornamento sarà anch'esso $O(d^{2n})$.

Per questi motivi, anche se i modelli HMM trovano applicazione in vari campi, dal riconoscimento vocale alla biologia molecolare, sono limitati nella capacità di rappresentare processi complessi. Nella terminologia introdotta nel Capitolo 2, i modelli HMM sono rappresentazioni atomiche: gli stati del mondo sono privi di struttura interna e sono etichettati con numeri interi. Nel Paragrafo 14.5 vedremo come usare le reti bayesiane dinamiche – una rappresentazione fattorizzata – per modellare domini con molte variabili di stato. Nel paragrafo seguente vedremo come gestire domini con variabili di stato continue, che naturalmente portano a uno spazio degli stati infinito.

14.4 Filtri di Kalman

Immaginate di osservare un uccellino che vola al tramonto attraverso il denso fogliame di una foresta: quelli che vedete sono brevi, intermittenti lampi di movimento; è difficile indovinare dov'è l'uccello e dove apparirà nell'istante successivo. O immaginate di essere un operatore radar durante la Seconda Guerra Mondiale, fissando un puntino di luce in movimento che appare sullo schermo una volta ogni 10 secondi. Oppure, andando ancor più indietro nel tempo, immaginate di essere Keplero che cerca di ricostruire il moto dei pianeti da una collezione di osservazioni angolari inaccurate prese a intervalli irregolari e misurati in modo impreciso.

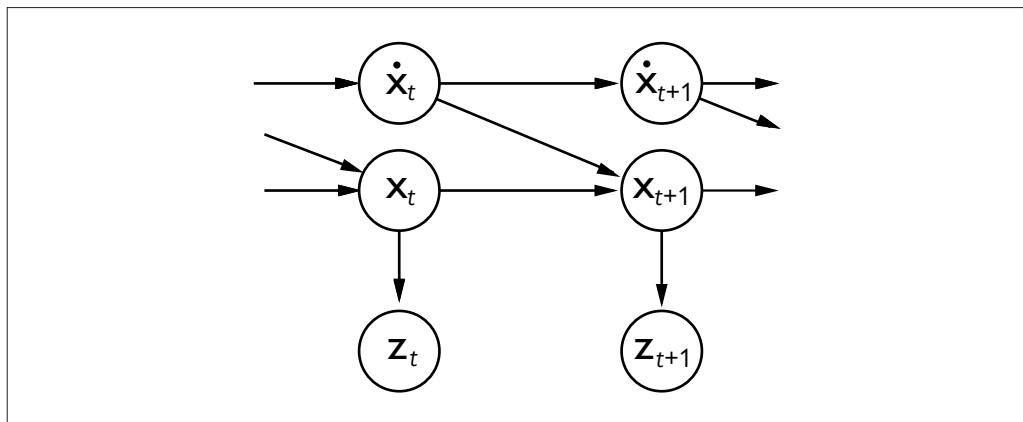
In tutti questi casi, state eseguendo un filtraggio: stimare variabili di stato (in questi esempi, la posizione e la velocità di un oggetto in movimento) a partire da osservazioni rumorese fatte nel tempo. Se le variabili fossero discrete, potremmo utilizzare un modello di Markov nascosto. In questo paragrafo esamineremo metodi per gestire variabili continue, usando un algoritmo denominato **filtraggio di Kalman** dal nome del suo inventore, Rudolf Kalman.

Il volo dell'uccellino potrebbe essere specificato da sei variabili continue in ogni istante temporale: tre per la posizione (X_t, Y_t, Z_t) e tre per la velocità ($\dot{X}_t, \dot{Y}_t, \dot{Z}_t$). Ci serviranno densità condizionali adeguate per rappresentare il modello di transizione e quello sensoriale; come nel Capitolo 13 useremo distribuzioni **gaussiane lineari**. Questo significa che lo stato successivo \mathbf{X}_{t+1} dev'essere una funzione lineare dello stato corrente \mathbf{X}_t più un rumore gaussiano, una condizione che nella pratica si dimostra abbastanza ragionevole. Considerate per esempio la coordinata X dell'uccello, ignorando per ora le altre. Chiamiamo Δ l'intervallo di tempo tra le osservazioni e assumiamo che la velocità di volo sia costante durante tale intervallo; allora l'aggiornamento della posizione è dato da $X_{t+\Delta} = X_t + \dot{X}\Delta$. Aggiungendo

filtraggio di Kalman

Figura 14.9

Struttura della rete bayesiana per un sistema lineare dinamico con posizione \mathbf{X}_t , velocità $\dot{\mathbf{X}}_t$ e misura della posizione \mathbf{Z}_t .



rumore gaussiano (per tenere conto di variazioni del vento e così via) otteniamo un modello di transizione gaussiano lineare:

$$P(X_{t+\Delta} = x_{t+\Delta} | X_t = x_t, \dot{X}_t = \dot{x}_t) = \mathcal{N}(x_{t+\Delta}; x_t + \dot{x}_t \Delta, \sigma^2).$$

La struttura della rete bayesiana per un sistema con vettore di posizione \mathbf{X}_t e velocità $\dot{\mathbf{X}}_t$ è mostrata nella Figura 14.9. Notate che questa è una forma molto particolare di modello gaussiano lineare; la forma generale, descritta più avanti in questo paragrafo, copre una vasta gamma di applicazioni che vanno ben oltre i semplici esempi che abbiamo proposto qui sopra. Il lettore potrebbe desiderare di consultare l'Appendice A per consolidare la sua conoscenza delle distribuzioni gaussiane; per i nostri scopi immediati, la proprietà più importante è che una distribuzione **gaussiana multivariata** con d variabili è specificata da una media a d elementi μ e una matrice $d \times d$ di covarianza Σ .

14.4.1 Aggiornare le distribuzioni gaussiane

Nel Capitolo 13 abbiamo accennato a una proprietà fondamentale della famiglia delle distribuzioni gaussiane lineari, e cioè la sua chiusura rispetto all'aggiornamento di reti bayesiane (questo significa che, data qualsiasi evidenza, la distribuzione a posteriori rientra ancora nella famiglia delle distribuzioni gaussiane lineari). Ora preciseremo questa affermazione nel contesto del filtraggio in un modello di probabilità temporale. Le proprietà richieste corrispondono al calcolo in due passi dell'Equazione (14.5).

1. Se la distribuzione corrente $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ è gaussiana e il modello di transizione $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t)$ è gaussiano lineare, anche la predizione della distribuzione a un passo data da

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) d\mathbf{x}_t \quad (14.17)$$

è anch'essa gaussiana.

2. Se la predizione $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})$ è gaussiana e il modello sensoriale $\mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})$ è gaussiano lineare, dopo il condizionamento sulle nuove evidenze la distribuzione aggiornata

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \quad (14.18)$$

è anch'essa gaussiana.

Così, l'operatore FORWARD per il filtraggio di Kalman prende un messaggio in avanti gaussiano $\mathbf{f}_{1:t}$, specificato dalla media μ_t e dalla covarianza Σ_t , e produce un nuovo messaggio in avanti gaussiano multivariato $\mathbf{f}_{1:t+1}$, specificato dalla media μ_{t+1} e dalla matrice di covarianza

Σ_{t+1} . Così, se partiamo da una distribuzione a priori gaussiana $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0) = \mathcal{N}(\mu_0, \Sigma_0)$, il filtraggio con un modello gaussiano lineare produce per sempre una distribuzione di stato gaussiana.

Questo è un risultato elegante ma per quale ragione è così importante? La ragione è che, tranne che in pochi casi speciali come questo, *il filtraggio con reti continue o ibride (discrete e continue) genera distribuzioni di stato la cui rappresentazione cresce senza limiti nel tempo*. Questa affermazione non è facile da provare nel caso generale, ma l'Esercizio 14.KFSW mostra cosa succede con un semplice esempio.



14.4.2 Un semplice esempio monodimensionale

Come abbiamo detto, l'operatore FORWARD del filtro di Kalman fa corrispondere a una gaussiana una nuova gaussiana. Questo significa calcolare una nuova media e una nuova covarianza da quelle precedenti. Derivare la regola di aggiornamento nel caso generale (multivariato) richiede una notevole quantità di algebra lineare, ragion per cui per adesso ci limiteremo a un esempio molto semplice univariato; più avanti forniremo i risultati per il caso generale. Anche nel caso univariato i calcoli sono alquanto lunghi e noiosi, ma pensiamo che valga la pena esaminarli con attenzione perché l'utilità del filtro di Kalman è intimamente collegata alle proprietà matematiche delle distribuzioni gaussiane.

Il modello temporale che consideriamo descrive la **camminata casuale** (*random walk*) di una singola variabile di stato continua X_t con un'osservazione rumorosa Z_t . Potremmo pensare che la variabile rappresenti un qualche indice di “fiducia dei clienti” che ogni mese subisce un cambiamento con distribuzione gaussiana, misurato con una ricerca di mercato che a sua volta introduce rumore gaussiano nel campionamento. Assumiamo che la distribuzione a priori sia una gaussiana con varianza σ_0^2 :

$$P(x_0) = \alpha e^{-\frac{1}{2}\left(\frac{(x_0 - \mu_0)^2}{\sigma_0^2}\right)}.$$

Notate che, per semplicità, in tutto questo paragrafo useremo lo stesso simbolo α per tutte le costanti di normalizzazione. Il modello di transizione si limita semplicemente ad aggiungere allo stato corrente una perturbazione gaussiana di varianza costante σ_x^2 :

$$P(x_{t+1} | x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(x_{t+1} - x_t)^2}{\sigma_x^2}\right)}.$$

Il modello sensoriale assume un rumore gaussiano con varianza σ_z^2 :

$$P(z_t | x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(z_t - x_t)^2}{\sigma_z^2}\right)}.$$

Ora, data la distribuzione $P(X_0)$, la predizione della distribuzione a un passo deriva dall'Equazione (14.17):

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1 | x_0) P(x_0) dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1 - x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0 - \mu_0)^2}{\sigma_0^2}\right)} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1 - x_0)^2 + \sigma_x^2(x_0 - \mu_0)^2}{\sigma_0^2 \sigma_x^2}\right)} dx_0. \end{aligned}$$

Questo integrale ha un aspetto piuttosto complicato. La chiave sta nel notare che l'esponente è la somma di due espressioni *quadratiche* in x_0 ed è quindi quadratico in x_0 anch'esso. Un semplice trucco, noto come **completare il quadrato**, ci permette di riscrivere qualsiasi espressione quadratica $ax_0^2 + bx_0 + c$ come la somma di un termine al quadrato $a(x_0 - \frac{-b}{2a})^2$ e di un

completare
il quadrato

termine residuo $c - \frac{b^2}{4a}$ indipendente da x_0 . In questo caso abbiamo $a = (\sigma_0^2 + \sigma_x^2)/(\sigma_0^2 \sigma_x^2)$, $b = -2(\sigma_0^2 x_1 + \sigma_x^2 \mu_0)/(\sigma_0^2 \sigma_x^2)$, e $c = (\sigma_0^2 x_1^2 + \sigma_x^2 \mu_0^2)/(\sigma_0^2 \sigma_x^2)$. Il termine residuo può essere portato fuori dall'integrale, ottenendo:

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(c - \frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a(x_0 - \frac{-b}{2a})^2\right)} dx_0.$$

Ora l'integrale non è altro che l'integrale di una gaussiana su tutto il suo dominio, che vale semplicemente 1: ci rimane così solo il termine residuo. Inserendo di nuovo le espressioni per a , b e c e semplificando, otteniamo:

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2}\right)}.$$

Ovvero, la distribuzione predetta a un passo è una gaussiana con la stessa media μ_0 e una varianza pari alla somma della varianza originale σ_0^2 e di quella di transizione σ_x^2 .

Per completare il passo di aggiornamento dobbiamo condizionare il nuovo stato sull'osservazione nel primo istante temporale, cioè z_1 . Dall'Equazione (14.18), risulta

$$\begin{aligned} P(x_1 | z_1) &= \alpha P(z_1 | x_1) P(x_1) \\ &= \alpha e^{-\frac{1}{2}\left(\frac{(z_1 - x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2}\right)}. \end{aligned}$$

Ancora una volta, combiniamo gli esponenti e completiamo il quadrato (cfr. Esercizio 14.KALM), ottenendo la seguente espressione per la distribuzione a posteriori:

$$P(x_1 | z_1) = \alpha e^{-\frac{1}{2}\left[\left(x_1 - \frac{(\sigma_0^2 + \sigma_x^2)z_1 + \sigma_z^2 \mu_0}{\sigma_0^2 + \sigma_x^2 + \sigma_z^2}\right)^2\right]} \quad (14.19)$$

Così, dopo un ciclo di aggiornamento, abbiamo una nuova distribuzione gaussiana per la variabile di stato.

Dalla formula della gaussiana nell'Equazione (14.19) vediamo che la nuova media e la nuova deviazione standard possono essere calcolate da quelle vecchie come segue:

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \quad \text{e} \quad \sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}. \quad (14.20)$$

La Figura 14.10 mostra un ciclo di aggiornamento per valori particolari dei modelli di transizione e sensoriale.

L'Equazione (14.20) ricopre esattamente lo stesso ruolo dell'equazione del filtraggio generale (14.5) o per HMM (14.12). Data la natura speciale delle distribuzioni gaussiane, comunque, le equazioni hanno alcune caratteristiche interessanti aggiuntive.

Per prima cosa possiamo interpretare il calcolo della nuova media μ_{t+1} come una *media pesata* della nuova osservazione z_{t+1} e della vecchia media μ_t . Se l'osservazione non è affidabile, allora σ_z^2 è grande e daremo più peso alla vecchia media; se invece è quest'ultima a essere inaffidabile (σ_t^2 è grande) o il processo è difficilmente predicibile (σ_x^2 è grande), daremo più peso all'osservazione. In secondo luogo, notate che l'aggiornamento della varianza σ_{t+1}^2 è *indipendente dall'osservazione*. Possiamo quindi calcolare in anticipo la sequenza di valori della varianza. Terzo, la sequenza di valori della varianza converge rapidamente a un valore fisso che dipende solo da σ_x^2 e σ_z^2 , cosa che rende decisamente più semplici i calcoli successivi (cfr. Esercizio 14.VARI).

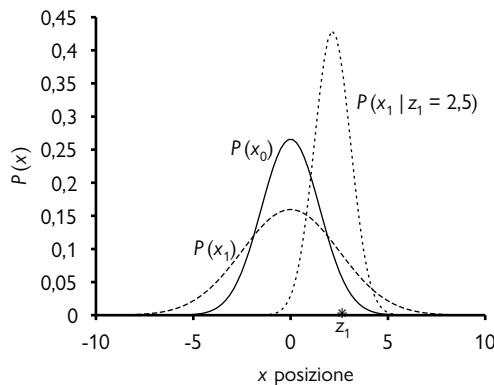


Figura 14.10 Fasi del ciclo di aggiornamento del filtro di Kalman per una camminata casuale con una probabilità a priori data da $\mu_0 = 0,0$ e $\sigma_0 = 1,5$, rumore di transizione dato da $\sigma_x = 2,0$, rumore dei sensori dato da $\sigma_z = 1,0$ e una prima osservazione $z_1 = 2,5$ (segnata sull'asse delle x). Notate come il rumore di transizione "appiattisce" la predizione $P(x_1)$ rispetto a $P(x_0)$. Notate anche che la media della probabilità a posteriori $P(x_1|z_1)$ è leggermente spostata a sinistra dell'osservazione z_1 , dato che è calcolata pesando la predizione e l'osservazione.

14.4.3 Il caso generale

La precedente derivazione illustra la caratteristica chiave delle distribuzioni gaussiane, che permette il funzionamento del filtraggio di Kalman: il fatto che l'esponente sia in forma quadratica. Questo è vero anche nel caso non univariato; la distribuzione gaussiana multivariata completa ha la forma:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha e^{-\frac{1}{2}((\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}))}.$$

Con qualche moltiplicazione vediamo che l'esponente è anch'esso una funzione quadratica dei valori x_i di \mathbf{x} . Quindi, il filtraggio conserva la natura gaussiana della distribuzione di stato.

Per prima cosa definiamo il modello temporale generale del filtraggio di Kalman. Sia il modello di transizione che quello sensoriale devono essere una trasformazione *lineare* con l'aggiunta di rumore gaussiano. Abbiamo così:

$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_{t+1}; \mathbf{F}\mathbf{x}_t, \boldsymbol{\Sigma}_x) \\ P(\mathbf{z}_t|\mathbf{x}_t) &= \mathcal{N}(\mathbf{z}_t; \mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_z), \end{aligned} \quad (14.21)$$

dove \mathbf{F} e $\boldsymbol{\Sigma}_x$ sono matrici che descrivono il modello lineare di transizione e la covarianza del rumore di transizione; \mathbf{H} e $\boldsymbol{\Sigma}_z$ sono le matrici corrispondenti per il modello sensoriale. Così le equazioni di aggiornamento per la media e la covarianza, in tutta la loro orribile bruttezza, sono:

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \boldsymbol{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)^{-1} \end{aligned} \quad (14.22)$$

dove $\mathbf{K}_{t+1} = (\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)\mathbf{H}^T(\mathbf{H}(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)\mathbf{H}^T + \boldsymbol{\Sigma}_z)^{-1}$ è la **matrice dei guadagni di Kalman**. Che lo crediate o no, queste equazioni si possono comprendere intuitivamente. Per esempio, considerate l'aggiornamento della stima della media dello stato $\boldsymbol{\mu}$. Il termine $\mathbf{F}\boldsymbol{\mu}_t$ è lo stato *predetto* in $t+1$, per cui $\mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ è l'osservazione *predetta*. Quindi, il termine $\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ rappresenta l'errore nell'osservazione predetta. Quest'ultimo è moltiplicato per \mathbf{K}_{t+1} per correggere lo stato predetto; quindi \mathbf{K}_{t+1} è la misura di *quanto prendiamo sul serio la nuova osservazione* in relazione alla predizione. Come nell'Equazione (14.20), vale anche la proprietà

matrice dei
guadagni di Kalman

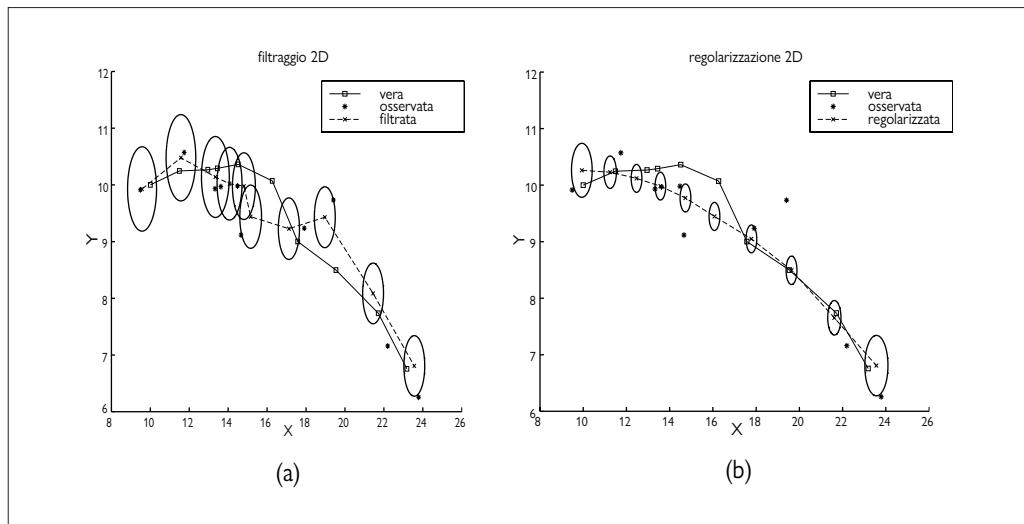


Figura 14.11 (a) Risultati del filtraggio di Kalman per un oggetto che si muove sul piano X–Y. È indicata la vera traiettoria (da sinistra a destra), una serie di osservazioni rumorose e la traiettoria stimata dal filtraggio. La varianza nella stima della posizione è indicata dagli ovali. (b) I risultati della regolarizzazione di Kalman per la stessa sequenza di osservazioni.

che l’aggiornamento della varianza è indipendente dalle osservazioni. La sequenza dei valori di Σ_t e \mathbf{K}_t può quindi essere calcolata offline, mentre i calcoli richiesti durante l’esecuzione sono molto modesti.

Per mostrare queste equazioni all’opera, le abbiamo applicate al problema del tracciamento di un oggetto che si muove su un piano X – Y . Le variabili di stato sono $\mathbf{X} = (X, Y, \dot{X}, \dot{Y})^\top$ cosicché \mathbf{F} , Σ_x , \mathbf{H} e Σ_z sono matrici 4×4 . La Figura 14.11(a) mostra la traiettoria reale, una serie di osservazioni rumorose e la traiettoria stimata dal filtro di Kalman, insieme con le covarianze indicate dagli ovali della deviazione standard. Il processo di filtraggio riesce a tracciare il movimento effettivo con buona approssimazione e, come ci aspettavamo, la varianza raggiunge presto un punto fisso.

Oltre che per il filtraggio, possiamo anche derivare equazioni per lo *smoothing* o regolarizzazione su modelli gaussiani lineari. I risultati sono mostrati nella Figura 14.11(b). Notate come la varianza sulla stima della posizione risulta nettamente ridotta, tranne che alla fine della traiettoria (perché?), e che la traiettoria stimata è molto più regolare (o liscia: “smooth”, appunto).

14.4.4 Applicabilità del filtraggio di Kalman

Il filtro di Kalman, con le sue varianti, è utilizzato in una vasta gamma di applicazioni: la più “classica” è il tracciamento radar di velivoli o missili. Un utilizzo analogo si ha nel tracciamento acustico di sommergibili e veicoli di terra, e in quello visuale di veicoli e persone. Un’applicazione più originale è la ricostruzione delle traiettorie delle particelle partendo da fotografie delle camere a bolle o quella delle correnti oceaniche sulla base delle misurazioni satellitari delle superfici. La versatilità del filtraggio di Kalman va ben oltre il semplice tracciamento dei movimenti: qualsiasi sistema caratterizzato da variabili di stato continue e misurazioni rumorose può giovarsene. Sistemi simili includono macchinari industriali, impianti chimici, reattori nucleari, ecosistemi botanici e persino le economie delle nazioni.

Il fatto che si possa applicare un filtro di Kalman a un sistema non significa che i risultati saranno validi o utili: le ipotesi che il modello di transizione e quello sensoriale siano gaussiani

lineari sono molto stringenti. Il **filtro di Kalman esteso (EKF, extended Kalman filter)** cerca di ovviare alle eventuali non-linearietà presenti. Un sistema è non lineare se il modello di transizione non può essere descritto da una moltiplicazione del vettore di stato per una matrice, come nell'Equazione (14.21). L'EKF considera il sistema come fosse *localmente* lineare in \mathbf{x}_t , nella regione di $\mathbf{x}_t = \mu_t$, la media della distribuzione di stato corrente. Questo funziona bene nel caso di sistemi regolari e dal comportamento uniforme, che permettono di mantenere e aggiornare una distribuzione gaussiana di stato che approssima ragionevolmente la distribuzione a posteriori reale. Un esempio dettagliato è fornito nel Capitolo 26 del Volume 2.

filtro di Kalman esteso (EKF)

Che cosa significa per un sistema essere “non regolare” o avere un comportamento “non uniforme”? Tecnicamente questo corrisponde a una significativa non-linearità nella risposta del sistema all’interno di una regione “vicina” (in base alla covarianza Σ_t) alla media corrente μ_t . Per comprendere quest’idea in termini non tecnici, considerate l’esempio del tracciamento del volo di un uccello attraverso una foresta, quando sembra che il volatile punti direttamente ad alta velocità contro il tronco di un albero. Il filtro di Kalman, regolare o esteso, può solo formulare una predizione gaussiana della posizione dell’uccello, e la media di tale gaussiana sarà per forza centrata sul tronco stesso, come si vede nella Figura 14.12(a). Un modello ragionevole, d’altra parte, dovrebbe prevedere un’azione evasiva su uno o l’altro dei lati dell’albero, come nella Figura 14.12(b). Un modello simile è senz’altro non-lineare, perché la decisione dell’animale varierà drasticamente in corrispondenza di piccole variazioni della sua posizione rispetto al tronco.

filtro di Kalman a commutazione

Per gestire esempi come questo è necessario disporre di un linguaggio più espressivo per rappresentare il comportamento del sistema modellato. Nella comunità della teoria del controllo, in cui problemi simili sono posti per esempio dalle manovre evasive dei velivoli militari, la soluzione standard è adottare un **filtro di Kalman a commutazione** (*switching Kalman filter*). Quest’approccio usa più filtri di Kalman in parallelo, ognuno con un modello diverso del sistema: per esempio uno per il volo diritto, uno per le brusche curve a sinistra e uno per quelle a destra. Si utilizza una somma pesata delle predizioni, in cui i pesi dipendono dal grado di corrispondenza di ogni filtro ai dati correnti. Nel prossimo paragrafo vedremo che questo è semplicemente un caso speciale del modello generale delle reti bayesiane dinami-

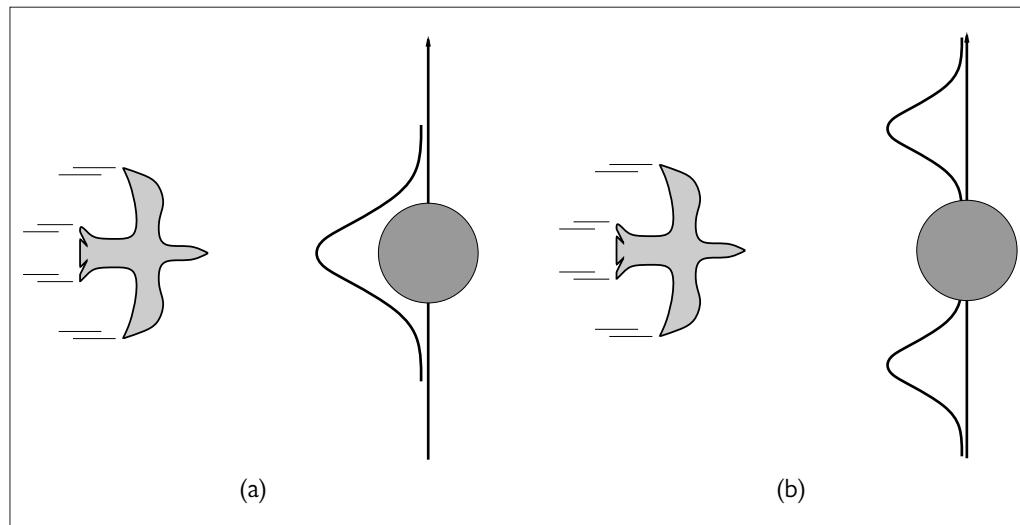


Figura 14.12 Un uccello che vola rapidamente contro il tronco di un albero (visto dall’alto). (a) Un filtro di Kalman cercherà di predire la posizione del volatile mediante una singola gaussiana centrata sull’ostacolo. (b) Un modello più realistico considera eventuali manovre evasive dell’uccello, predicendo che eviterà l’albero volando da una o dall’altra parte.

che, ottenute aggiungendo una variabile discreta “di manovra” alla rete della Figura 14.9. I filtri di Kalman a commutazione sono ulteriormente discussi nell’Esercizio 14.KFSW.

14.5 Reti bayesiane dinamiche

rete bayesiana
dinamica

Le **reti bayesiane dinamiche**, o **DBN** (*dynamic Bayesian network*), estendono la semantica delle reti bayesiane standard per gestire modelli temporali probabilistici del tipo descritto nel Paragrafo 14.1. Abbiamo già visto alcuni esempi di DBN: la rete dell’ombrellino nella Figura 14.2 e quella del filtro di Kalman nella Figura 14.9. In generale, ogni time slice di una DBN può avere un numero arbitrario di variabili di stato \mathbf{X}_t e di variabili di evidenza \mathbf{E}_t . Per semplicità assumeremo che le variabili, i loro collegamenti e le loro distribuzioni condizionate siano replicate esattamente da una time slice all’altra e che la DBN rappresenti un processo di Markov del primo ordine, in modo che ogni variabile possa avere genitori solo nella sua time slice e in quella immediatamente precedente. In questo modo, la DBN corrisponde a una rete bayesiana con infinite variabili.

Dovrebbe esservi chiaro che ogni modello di Markov nascosto può essere rappresentato da una DBN con una sola variabile di stato e una di evidenza. Inoltre ogni DBN a variabili discrete può essere rappresentata come un HMM; come abbiamo detto nel Paragrafo 14.3, basta combinare tutte le variabili di stato della DBN in una “megavariabile” i cui valori corrispondono a tutte le possibili tuple di valori delle singole variabili. Ora, se ogni HMM è una DBN e ogni DBN può essere trasformata in un HMM, dov’è la differenza? La differenza sta nel fatto che, *scomponendo lo stato di un sistema complesso nelle sue variabili costitutive, possiamo trarre vantaggio dalla sparsità del modello temporale probabilistico*.

Per capire che cosa significa questo nella pratica, ricordate che nel Paragrafo 14.3 abbiamo detto che una rappresentazione HMM per un processo temporale con n variabili discrete, ognuna delle quali può avere fino a d valori, richiede una matrice di transizione di dimensione $O(d^{2n})$. La rappresentazione DBN, invece, ha dimensione $O(nd^k)$ se il numero di genitori di ogni variabile è limitato da k . In altre parole, la rappresentazione DBN è lineare anziché esponenziale nel numero di variabili. Per il robot aspirapolvere con 42 possibili posizioni sporche, il numero di valori di probabilità richiesto si riduce da 5×10^{29} a poche migliaia.

Abbiamo già visto che ogni modello di filtro di Kalman può essere rappresentato come una DBN con variabili continue e distribuzioni condizionate gaussiane lineari (Figura 14.9). Grazie alla discussione che ha concluso il paragrafo precedente, dovrebbe ora essere chiaro che *non* tutte le DBN possono essere rappresentate da un modello di filtro di Kalman. In quest’ultimo, infatti, la distribuzione dello stato corrente è sempre una singola gaussiana multivariata: informalmente, una “gobba” in una particolare posizione dello spazio degli stati. Le DBN, d’altro canto, possono modellare distribuzioni arbitrarie.

In molte applicazioni reali, questa flessibilità è fondamentale. Considerate per esempio la posizione corrente delle mie chiavi di casa. Potrebbero essere nella tasca della mia giacca, sul comodino accanto al letto, sul piano della cucina, infilate nella toppa della porta, o chiuse nell’auto. Una singola curva gaussiana che includa tutti questi posti dovrebbe assegnare una probabilità significativa alla possibilità che le chiavi si trovino a mezz’aria in giardino. Aspetti del mondo reale come agenti dotati di volontà, ostacoli o tasche introducono delle “non linearità” che richiedono una combinazione di variabili discrete e continue per ottenere modelli ragionevoli.

14.5.1 Costruire le DBN

Per costruire una DBN occorre specificare tre tipi di informazioni: la distribuzione a priori delle variabili di stato, $\mathbf{P}(\mathbf{X}_0)$, il modello di transizione $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t)$ e quello sensoriale $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$. Per specificare il modello di transizione e quello sensoriale si deve anche specificare



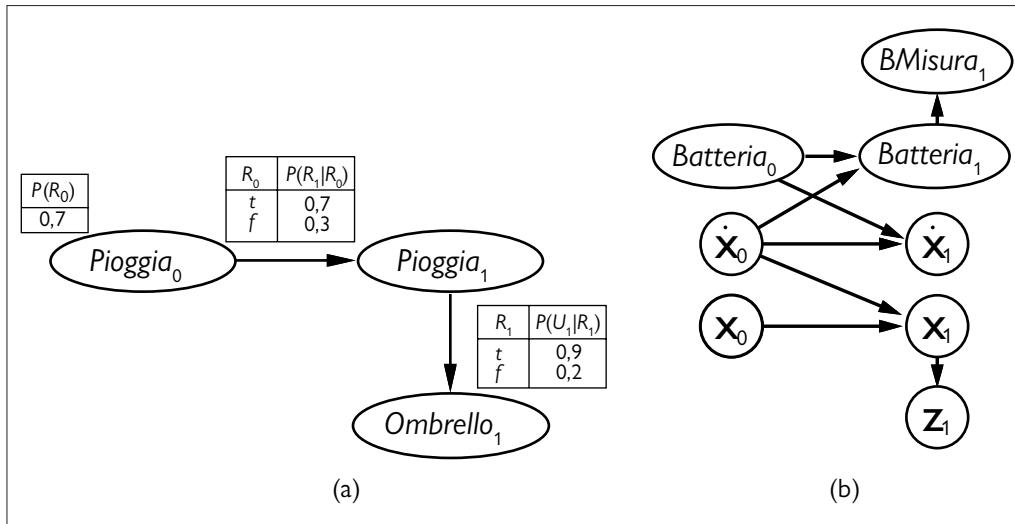


Figura 14.13 A sinistra: specifica del modello a priori, di transizione e sensoriale per la DBN dell'ombrellino. Le time slice successive sono semplici copie della prima. A destra: una semplice DBN per il movimento di un robot su un piano X-Y.

la topologia delle connessioni tra istanti temporali successivi e tra le variabili di stato e quelle di evidenza. Dato che si assume che il modello di transizione e quello sensoriale siano omogenei rispetto al tempo (ovvero identici in ogni istante t) è più comodo limitarsi a specificarli per la prima time slice. Per esempio, la specifica DBN completa del mondo dell'ombrellino è data dalla rete a tre nodi riportata nella Figura 14.13(a). Partendo da questa, la DBN completa con un numero illimitato di time slice può essere costruita, quando necessario, semplicemente copiando la prima time slice.

Consideriamo ora un esempio più interessante: il monitoraggio di un robot alimentato a batteria che si muove su un piano X-Y, presentato alla fine del Paragrafo 14.1. Per prima cosa dobbiamo definire le variabili di stato, che includeranno sia la posizione $\mathbf{X}_t = (X_t, Y_t)$ che la velocità $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$. Assumiamo che esista un qualche metodo per misurare la posizione: forse una telecamera fissa, o un sistema GPS (*global positioning system*); in ogni caso il dispositivo restituirà una misura \mathbf{Z}_t . La posizione nell'istante temporale successivo dipende dalla posizione e dalla velocità correnti, come nel modello standard del filtro di Kalman. La velocità nel passo successivo dipende dalla velocità corrente e dallo stato della batteria. Aggiungiamo $Batteria_t$ per rappresentare l'effettivo livello di carica, che ha come genitori il livello di carica precedente e la velocità, nonché $BMisura_t$, la misura del livello stesso. Tutto questo ci dà il modello della Figura 14.13(b).

Vale la pena di considerare con qualche attenzione la natura del modello sensoriale di $BMisura_t$. Supponiamo per semplicità che sia $Batteria_t$ che $BMisura_t$ possano assumere i valori discreti da 0 a 5 (l'Esercizio 14.BATT vi chiede di mettere in relazione questo modello discreto con un modello continuo corrispondente). Se il sensore è sempre accurato, la tabella delle probabilità condizionate $\mathbf{P}(BMisura_t|Batteria_t)$ dovrebbe avere probabilità 1,0 “lungo la diagonale” e 0,0 altrove. In realtà, nessuna misurazione è immune dal rumore. Nel caso di misure continue, si potrebbe usare una distribuzione gaussiana con una piccola varianza;⁷

⁷ In senso stretto, la distribuzione gaussiana è problematica perché assegna una probabilità maggiore di zero a livelli di carica negativi. Nel caso di variabili dal dominio ristretto, talvolta è una buona idea usare al suo posto una **distribuzione beta**.

modello di errore gaussiano

per le nostre variabili discrete possiamo approssimare la gaussiana con una distribuzione in cui la probabilità di errore scema nel modo appropriato, in modo tale che la probabilità di allontanarsi parecchio dal valore reale sia molto piccola. Useremo il termine **modello di errore gaussiano** per riferirci sia al caso continuo che a quello discreto.

fallimento transitorio

Chiunque abbia esperienza diretta di robotica, controllo di processo computerizzato o altre forme di percezione automatica sarà pronto a testimoniare che spesso la presenza di piccoli errori nella misurazione è l'ultimo dei problemi. Il fatto è che i sensori reali *falliscono*. Quando un sensore fallisce, non invia necessariamente un segnale che avverte “oh, a proposito, i dati che sto per inviare non hanno alcun senso”: si limita semplicemente a inviare i dati privi di senso. Il tipo più semplice è il **fallimento transitorio**, il cui il sensore decide occasionalmente di inviare dati senza significato. Per esempio, il sensore di livello della batteria potrebbe avere il difetto di inviare una lettura uguale a zero quando qualcuno urta il robot, anche se la batteria è completamente carica.

Vediamo cosa accade quando si verifica un fallimento transitorio con un modello di errore gaussiano che non è in grado di gestirlo. Supponiamo, per esempio, che il robot sia fermo e tranquillo e osservi 20 letture consecutive di 5 come livello della batteria. A questo punto il sensore ha un piccolo raptus e la lettura successiva è $B\text{Misura}_{21} = 0$. Cosa ci porterà a credere il nostro semplice modello di errore gaussiano riguardo a $Batteria_{21}$? Secondo la regola di Bayes la risposta dipende sia dal modello sensoriale $\mathbf{P}(B\text{Misura}_{21} = 0 | Batteria_{21})$ che dalla predizione $\mathbf{P}(Batteria_{21} | B\text{Misura}_{1:20})$. Se la probabilità di un grande errore del sensore è significativamente inferiore a quella di una transizione a $Batteria_{21} = 0$, anche se quest'ultima è molto improbabile, la distribuzione a posteriori assegnerà un'alta probabilità che la batteria si sia completamente scaricata.

Una seconda lettura di 0 nell'istante $t = 22$ renderà quasi certa questa conclusione. Se il fallimento transitorio dovesse successivamente scomparire, e la lettura del sensore tornasse al valore 5 da $t = 23$ in poi, la stima del livello della batteria tornerebbe rapidamente a 5 (questo non significa che l'algoritmo pensi che la batteria si sia ricaricata da sola come per magia, il che potrebbe essere fisicamente impossibile; invece, l'algoritmo ora pensa che il livello della batteria non è mai stato basso e che l'ipotesi estremamente improbabile che il misuratore del livello di carica presenti due grandi errori consecutivi debba essere la spiegazione corretta). Il corso degli eventi è illustrato dalla curva superiore nella Figura 14.14(a), che riporta il valore atteso di $Batteria_t$ nel tempo (cfr. l'Appendice A) utilizzando un modello di errore gaussiano discreto.

**modello di fallimento transitorio**

Nonostante il pronto recupero, c'è un istante ($t = 22$) in cui il robot è convinto che la sua batteria sia scarica; presumibilmente in questo caso dovrebbe inviare un segnale di allarme e spegnersi. Ahinoi, il suo modello sensoriale troppo semplificato l'ha indotto in errore. La morale della storia è semplice: *affinché il sistema possa gestire in modo corretto il fallimento dei sensori, il modello sensoriale deve includere la possibilità del fallimento*.

Il più semplice modello di fallimento per un sensore assegna una certa probabilità alla lettura di un valore completamente sbagliato, indipendentemente dallo stato del mondo. Per esempio, se l'indicatore della batteria fallisce restituendo 0, potremmo dire che:

$$\mathbf{P}(B\text{Misura}_t = 0 | Batteria_t = 5) = 0,03,$$

che è presumibilmente molto più grande della probabilità che potrebbe essere assegnata da un semplice modello di errore gaussiano. Chiamiamo questo **modello di fallimento transitorio**. In che modo questo ci aiuta quando riceviamo una lettura pari a 0? Posto che la *predizione* della probabilità di una batteria completamente scarica, secondo le letture ottenute fin qui, è molto inferiore a 0,03, la spiegazione migliore dell'osservazione $B\text{Misura}_{21} = 0$ è che il sensore abbia temporaneamente fallito. Intuitivamente, possiamo pensare che la credenza riguardante il livello della batteria abbia una certa quantità di “inerzia” che ci aiuta a superare fluttuazioni momentanee nella lettura dei valori. La curva superiore nella Figura

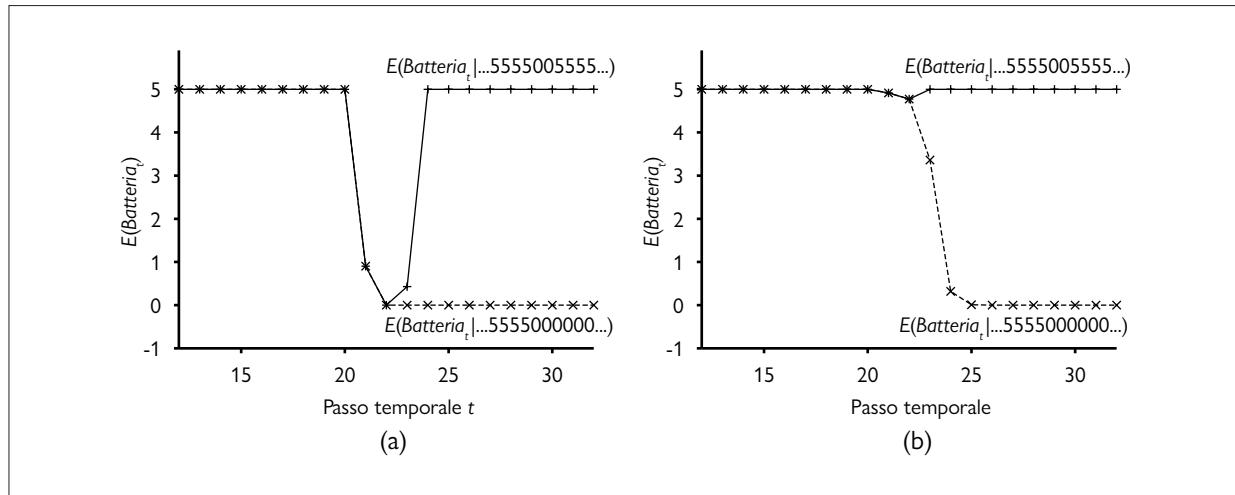


Figura 14.14 (a) Curva superiore: traiettoria del valore atteso di $Batteria_t$ per una sequenza di osservazioni che consiste in tutti 5 tranne due zeri in $t = 21$ e $t = 22$, utilizzando un semplice modello di errore gaussiano. Curva inferiore: traiettoria quando le osservazioni rimangono a 0 da $t = 21$ in poi. (b) Lo stesso esperimento in presenza di un modello di fallimento transitorio. L'errore transitorio è gestito bene, mentre quello persistente ha come risultato un eccessivo pessimismo sulla carica della batteria.

14.14(b) mostra che il modello di fallimento transitorio può gestire situazioni simili senza un catastrofico cambiamento di credenze.

Questo per quanto riguarda le fluttuazioni temporanee: ma che dire del guasto permanente di un sensore? Purtroppo, fallimenti di questo tipo si verificano frequentemente. Se il sensore restituisce 20 letture di 5 seguite da 20 letture di 0, il modello di fallimento transitorio descritto qui sopra farà sì che il robot si convinca gradualmente che la batteria è scarica, quando in effetti potrebbe ben darsi che si sia rotto il sensore. La curva inferiore nella Figura 14.14(b) mostra la “traiettoria” della credenza in questo caso. Nell’istante $t = 25$, dopo cinque letture di 0, il robot si è convinto che la batteria è scarica. Naturalmente, se è effettivamente più probabile che si sia rotto il sensore, preferiremmo che il robot arrivi a quest’ultima conclusione.

Per gestire i fallimenti persistenti, non vi sorprenderà sapere che è necessario un **modello di fallimento persistente** che descrive il funzionamento dei sensori in condizioni normali e dopo un guasto. Per questo dobbiamo arricchire lo stato del sistema con una variabile aggiuntiva $SBRotto$, che descrive lo stato del sensore della batteria. La persistenza di un fallimento è modellata da un arco che collega $SBRotto_0$ a $SBRotto_1$. Questo **arco di persistenza** ha una CPT che in ogni passo temporale assegna una piccola probabilità di errore, poniamo 0,001, ma specifica anche che, una volta guasto, il sensore rimane tale. Quando il sensore è OK, il modello sensoriale di $BMisura$ è identico al modello di fallimento transitorio; quando è guasto $BMisura$ varrà sempre 0, indipendentemente dall’effettivo livello di carica della batteria.

Il modello di fallimento persistente per il sensore della batteria è riportato nella Figura 14.15(a). La Figura 14.15(b) mostra la sua prestazione su due sequenze di dati (fluttuazione temporanea e guasto permanente). Ci sono molte cose da notare riguardo a queste curve. Prima di tutto, in caso di errore temporaneo, la probabilità che il sensore sia rotto aumenta significativamente dopo la seconda lettura di 0, ma torna immediatamente a zero non appena si osserva un 5. In secondo luogo, in caso di fallimento persistente, la probabilità che il sensore sia rotto sale rapidamente fin quasi a 1 e non muta più. Infine, una volta appurato che il sensore si è guastato, il robot può solo presumere che la sua batteria si stia scaricando a un ritmo “normale”, come si vede dalla lenta discesa del livello di $E(Batteria_t | \dots)$.

**modello di
fallimento
persistente**

arco di persistenza

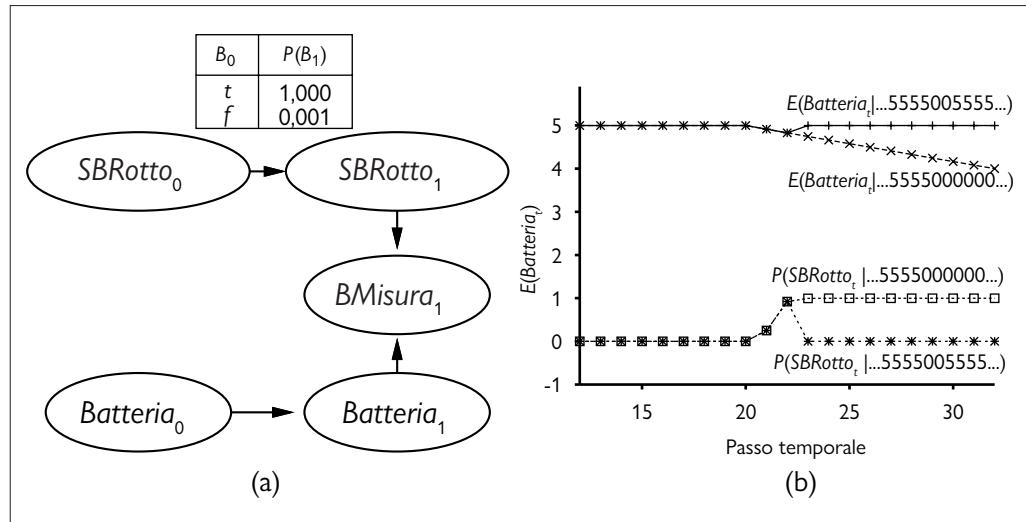


Figura 14.15 (a) Un frammento di DBN che illustra il funzionamento della variabile di stato necessaria per modellare il fallimento persistente del sensore della batteria. (b) Curve superiori: traiettorie del valore atteso di $Batteria_t$ per le sequenze di osservazioni “fallimento transitorio” e “fallimento permanente”. Curve inferiori: le traiettorie della probabilità di $SBRotto$ date le due sequenze di osservazioni.

Fin qui ci siamo fermati alla superficie del problema della rappresentazione di processi complessi. La varietà dei modelli di transizione è enorme e abbraccia campi disparati che vanno dalla modellazione del sistema endocrino umano a quella di un gruppo di veicoli in corsa su un’autostrada. La modellazione dei sensori è un sottocampo vasto in sé, ma le reti bayesiane dinamiche possono modellare anche fenomeni sottili come una taratura che varia lentamente, improvvise variazioni della calibrazione o gli effetti di condizioni esogene (come quelle atmosferiche) sulle letture dei sensori.

14.5.2 Inferenza esatta nelle DBN

Avendo delineato alcune idee relative alla rappresentazione di processi complessi sotto forma di DBN, occupiamoci ora della questione dell’inferenza. In un certo senso, abbiamo già dato una risposta a questo problema: le reti bayesiane dinamiche *sono* reti bayesiane, per le quali abbiamo già visto diversi algoritmi. Data una sequenza di osservazioni si può costruire la rappresentazione completa di una DBN in forma di rete bayesiana replicando le time slice finché la rete non è abbastanza grande da considerare tutte le osservazioni, come si vede nella Figura 14.16. Questa tecnica viene chiamata **srotolamento** (*unrolling*). Tecnicamente la DBN è equivalente alla rete semi-infinita ottenuta proseguendo lo srotolamento all’infinito; tutte le time slice successive all’ultima osservazione, tuttavia, non hanno alcun effetto sull’inferenza all’interno del periodo considerato e possono quindi essere omesse. Una volta che la DBN è srotolata, si può usare uno qualsiasi degli algoritmi di inferenza descritti nel Capitolo 13: eliminazione delle variabili, metodi di clustering e così via.

Sfortunatamente, un’applicazione ingenua dello srotolamento non risulta particolarmente efficiente. Se volessimo eseguire un filtraggio o una regolarizzazione su una lunga sequenza di osservazioni $e_{1:t}$, la rete srotolata richiederà uno spazio $O(t)$ e crescerà quindi senza limiti man mano che si aggiungono altre osservazioni. Inoltre, se ci limitassimo semplicemente a eseguire da zero l’algoritmo di inferenza ogni volta che aggiungiamo un’osservazione, anche il tempo richiesto per ogni aggiornamento crescerà con $O(t)$.

Ritornando al Paragrafo 14.2.1, possiamo vedere che è possibile ottenere requisiti spaziali e temporali costanti per ogni aggiornamento, a patto di riuscire a eseguire i calcoli in modo

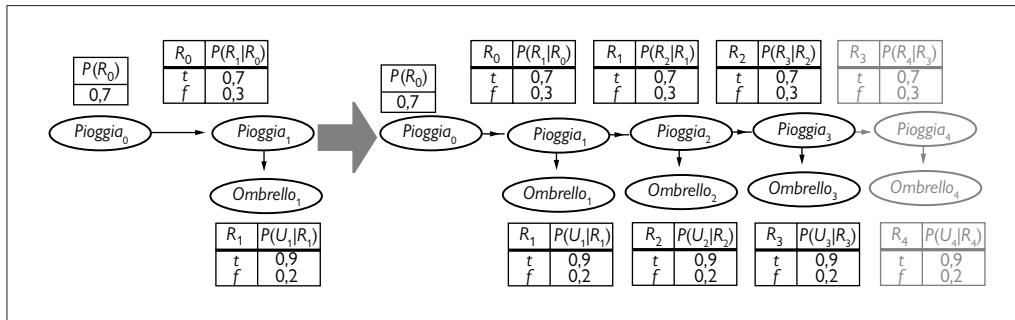


Figura 14.16 Srotolamento di una rete bayesiana dinamica: le time slice sono replicate in numero sufficiente a considerare l'intera sequenza di osservazioni $Ombrello_{1:3}$. Gli istanti successivi non hanno alcun effetto sull'inferenza all'interno del periodo osservato.

ricorsivo. Essenzialmente, l'aggiornamento di filtraggio nell'Equazione (14.5) funziona *eliminando attraverso una somma* le variabili di stato del passo precedente per ottenere la distribuzione nel nuovo istante. Il “summing out” delle variabili è esattamente l'operazione svolta dall'algoritmo di **eliminazione delle variabili** (Figura 13.13), e in effetti l'esecuzione di quest'ultimo con le variabili in ordine temporale ricalca esattamente il funzionamento dell'aggiornamento ricorsivo dell'Equazione (14.5). L'algoritmo modificato tiene in memoria in ogni momento al più due time slice: partendo dall'istante 0, aggiungiamo l'istante 1, quindi eliminiamo l'istante 0 mediante una somma, poi aggiungiamo l'istante 2, poi eliminiamo l'istante 1, e così via. In questo modo possiamo effettuare aggiornamenti di filtraggio con un tempo e uno spazio costanti (lo stesso risultato si può ottenere modificando in modo adeguato l'algoritmo di clustering). L'Esercizio 14.DBNE vi chiederà di verificare ciò sulla rete del mondo dell'ombrello.

Queste erano le buone notizie; ecco ora quelle cattive: la complessità “costante” spaziale e temporale per gli aggiornamenti è, in quasi tutti i casi, esponenziale nel numero delle variabili di stato. Man mano che l'eliminazione delle variabili procede, i fattori crescono fino a includere tutte le variabili di stato (o, per essere più precisi, tutte quelle che hanno genitori nell'istante precedente). La dimensione massima dei fattori è $O(d^{n+k})$ e il costo di aggiornamento totale per passo è $O(nd^{n+k})$, dove d è la dimensione del dominio della variabile e k è il numero massimo di genitori di ogni variabile di stato.

Naturalmente questo è sempre molto inferiore al costo di aggiornamento di un HMM, che è $O(d^{2n})$, ma rimane comunque inapplicabile quando il numero di variabili è grande. È difficile da accettare, ma *anche se possiamo usare le DBN per rappresentare processi temporali molto complessi con molte variabili sparsamente connesse, non possiamo ragionare su tali processi in modo efficiente ed esatto*. Il modello stesso delle DBN, che rappresenta la distribuzione congiunta a priori su tutte le variabili, è fattorizzabile nelle CPT che lo costituiscono, ma la distribuzione congiunta a posteriori condizionata alla sequenza di osservazioni – ovvero il messaggio in avanti – generalmente *non* lo è. Il problema in generale è intrattabile, perciò dobbiamo ripiegare su metodi approssimati.



14.5.3 Inferenza approssimata nelle DBN

Nel Paragrafo 13.4 abbiamo descritto due algoritmi approssimati: la pesatura di verosimiglianza (Figura 13.18) e gli algoritmi Monte Carlo per catene di Markov (Figura 13.20). Dei due, il primo è quello che si adatta più facilmente alle DBN (un algoritmo di filtraggio MCMC è descritto brevemente nelle note storiche e bibliografiche al termine di questo capitolo). Come vedremo, comunque, per ottenere un metodo applicabile in pratica sarà necessario apportare diverse migliorie all'algoritmo standard.

Ricorderete che la pesatura di verosimiglianza funziona campionando i nodi non di evidenza della rete in ordine topologico, pesando ogni campione in base alla verosimiglianza rispetto alle variabili di evidenza osservate. Come nel caso degli algoritmi esatti, potremmo applicare direttamente la pesatura di verosimiglianza a una DBN srotolata, ma ancora una volta i requisiti spaziali e temporali di ogni aggiornamento crescerebbero con le dimensioni della sequenza di osservazioni. Il problema è che l'algoritmo standard prende un campione per volta, attraversando tutta la rete.

È possibile considerare, invece, tutti gli N campioni, facendoli passare attraverso la DBN una time slice per volta. L'algoritmo modificato corrisponde allo schema generale degli algoritmi di filtraggio, con l'insieme di N campioni come messaggio in avanti. La prima miglioria fondamentale, quindi, è *usare gli stessi campioni come rappresentazione approssimata della distribuzione di stato corrente*. Questo soddisfa il requisito di un tempo di aggiornamento “costante”, benché tale costante dipenda dal numero di campioni necessari per mantenere un'approssimazione accurata della vera distribuzione. Inoltre non è necessario srotolare la DBN, dato che è sufficiente avere in memoria la time slice corrente e quella successiva. Questo approccio è chiamato **campionamento di importanza sequenziale**, o SIS (*sequential importance sampling*).

Nella discussione della pesatura di verosimiglianza svolta nel Capitolo 13, abbiamo notato che l'algoritmo risulta meno accurato quando le variabili di evidenza sono “a valle” di quelle che si stanno campionando, perché in tal caso i campioni vengono generati senza alcuna influenza da parte delle evidenze.

Ora, guardando la struttura tipica di una DBN (per esempio quella dell'ombrellino, nella Figura 14.16) vediamo che le prime variabili di stato saranno campionate senza l'apporto benefico delle successive evidenze. In effetti, guardando attentamente, *nessuna* variabile di stato ha *una sola* variabile di evidenza tra i suoi antenati! Di conseguenza, benché il peso di ogni campione continui a dipendere dalle evidenze, l'insieme di campioni generati sarà *completamente indipendente* dalle evidenze stesse. Così, anche se il capo dovesse continuare a portare ogni giorno l'ombrellino, il processo di campionamento potrebbe ancora generare (come in una allucinazione) una sequenza senza fine di giorni soleggiati.

Il significato pratico è che la frazione di campioni che rimangono ragionevolmente vicini all'effettiva serie degli eventi (e che, quindi, hanno pesi non trascurabili) decresce esponenzialmente con t , la lunghezza della sequenza. In altre parole, per mantenere un dato livello di accuratezza dovremo aumentare il numero di campioni esponenzialmente con t . Dato che un algoritmo di filtraggio in tempo reale può utilizzare solo un numero limitato di campioni, quello che accade in pratica è che l'errore esplode dopo un numero molto piccolo di passi di aggiornamento. La Figura 14.19, più avanti in questo capitolo, mostra questo effetto per il SIS applicato al problema di localizzazione del Paragrafo 14.3: anche con 100.000 campioni, l'approssimazione SIS fallisce completamente dopo circa 20 passi.

È chiaro che ci serve una soluzione migliore. La seconda innovazione consiste nel *focalizzare l'insieme dei campioni nelle regioni dello spazio degli stati ad alta probabilità*. Per far questo si possono gettar via i campioni che hanno un peso molto basso in base alle osservazioni, replicando invece quelli con un peso alto. In questo modo la popolazione di campioni rimarrà ragionevolmente vicina alla realtà. Se consideriamo i campioni come una risorsa per la modellazione della distribuzione a posteriori, è sensato utilizzarne di più nelle regioni dello spazio degli stati in cui essa è più alta.

Esiste una famiglia di algoritmi, chiamata **particle filtering** (inizialmente si utilizzava anche il termine **campionamento di importanza sequenziale con ricampionamento**, ma per qualche motivo è stato abbandonato), progettata per fare precisamente ciò. Il particle filtering funziona così: per prima cosa viene creata una popolazione di N campioni partendo dalla distribuzione a priori $\mathbf{P}(\mathbf{X}_0)$. Fatto questo si ripete a ogni passo temporale lo stesso ciclo di aggiornamento:



**campionamento
di importanza
sequenziale**



particle filtering

```
function PARTICLE-FILTERING(e,  $N$ ,  $dbn$ ) returns
```

un insieme di campioni per il passo temporale successivo

inputs: **e**, la nuova evidenza in ingresso

N , il numero di campioni da conservare

dbn , una DBN definita da $\mathbf{P}(\mathbf{X}_0)$, $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$ e $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$

persistent: S , un vettore di campioni di dimensione N , inizialmente generati da $\mathbf{P}(\mathbf{X}_0)$

local variables: W , un vettore di pesi di dimensione N

```
for  $i = 1$  to  $N$  do
```

$S[i] \leftarrow$ un campione da $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0 = S[i])$ // passo 1

$W[i] \leftarrow \mathbf{P}(\mathbf{e}| \mathbf{X}_1 = S[i])$ // passo 2

$S \leftarrow \text{CAMPIONAMENTO-PESATO-CON-RIMPIAZZO}(N, S, W)$ // passo 3

```
return  $S$ 
```

Figura 14.17 L'algoritmo di particle filtering, implementato come un'operazione ricorsiva di aggiornamento dotata di stato (l'insieme di campioni). Ogni passo richiede il campionamento delle variabili rilevanti per la time slice corrente in ordine topologico, in modo analogo a CAMPIONA-PRIORI. L'operazione CAMPIONAMENTO-PESATO-CON-RIMPIAZZO può essere implementata in modo da richiedere un tempo di esecuzione atteso $O(N)$. I numeri dei passi fanno riferimento alla descrizione riportata nel testo.

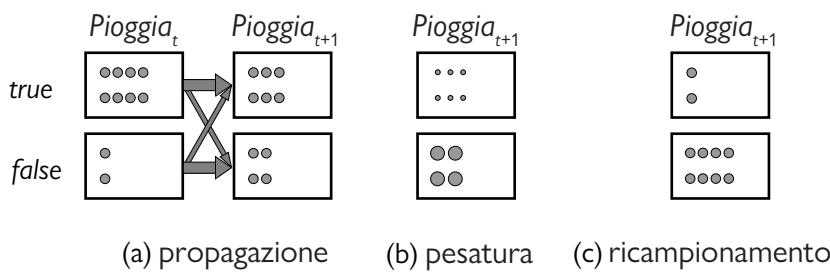


Figura 14.18 Il ciclo di aggiornamento del particle filtering per la DBN dell'ombrellino con $N = 10$, con le popolazioni di campioni in ogni stato. (a) All'istante t , 8 campioni indicano pioggia e 2 indicano \neg pioggia. Ognuno è propagato in avanti campionando lo stato successivo attraverso il modello di transizione. Al tempo $t + 1$, 6 campioni indicano pioggia e 4 indicano \neg pioggia. (b) In $t + 1$ si osserva \neg ombrello. Ogni campione viene pesato in base alla sua verosimiglianza rispetto all'osservazione, indicata in figura dalla dimensione dei cerchietti. (c) Un nuovo insieme di 10 campioni è generato mediante una selezione casuale pesata dall'insieme corrente, che ha come risultato 2 campioni che indicano pioggia e 8 che indicano \neg pioggia.

1. ogni campione è propagato in avanti campionando il valore di stato successivo \mathbf{x}_{t+1} dato quello corrente \mathbf{x}_t , in base al modello di transizione $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$;
2. ogni campione è pesato in base alla verosimiglianza che assegna alla nuova evidenza, $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$;
3. la popolazione è *ricampionata* per generare una nuova popolazione di N campioni. Ogni nuovo campione è preso dalla popolazione corrente; la probabilità di essere selezionato è proporzionale al suo peso. I nuovi campioni non sono pesati.

L'algoritmo è illustrato in dettaglio nella Figura 14.17, mentre la Figura 14.18 mostra il suo funzionamento sulla DBN dell'ombrellino.

Possiamo dimostrare che questo algoritmo è consistente (cioè fornisce le probabilità corrette quando N tende a infinito) esaminando che cosa accade durante un ciclo di aggiorna-

mento. Assumiamo che la popolazione dei campioni iniziale sia una rappresentazione corretta del messaggio in avanti, cioè che $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ al tempo t . Scrivendo $N(\mathbf{x}_t|\mathbf{e}_{1:t})$ per il numero di campioni che occupano lo stato \mathbf{x}_t dopo che sono state elaborate le osservazioni $\mathbf{e}_{1:t}$, abbiamo:

$$N(\mathbf{x}_t|\mathbf{e}_{1:t})/N = P(\mathbf{x}_t|\mathbf{e}_{1:t}) \quad (14.23)$$

per N grandi. Ora propaghiamo ogni campione in avanti campionando le variabili di stato al tempo $t + 1$, dati i valori del campione in t . Il numero di campioni che raggiungono lo stato \mathbf{x}_{t+1} da ogni \mathbf{x}_t corrisponde alla probabilità di transizione moltiplicata per la popolazione di \mathbf{x}_t ; il numero totale sarà quindi:

$$N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t) N(\mathbf{x}_t|\mathbf{e}_{1:t}).$$

Ora pesiamo ogni campione con la sua verosimiglianza rispetto all'evidenza al tempo $t + 1$. Un campione nello stato \mathbf{x}_{t+1} riceve un peso $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$. Il peso totale dei campioni in \mathbf{x}_{t+1} dopo aver considerato \mathbf{e}_{t+1} è quindi:

$$W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}).$$

Infine, il passo di ricampionamento. Dato che ogni campione è replicato con una probabilità proporzionale al suo peso, il numero di campioni nello stato \mathbf{x}_{t+1} dopo il ricampionamento è proporzionale al peso totale in \mathbf{x}_{t+1} prima di esso:

$$\begin{aligned} N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1})/N &= \alpha W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) \\ &= \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) \\ &= \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})\sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)N(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha NP(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})\sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t}) \quad (\text{per la 14.23}) \\ &= \alpha' P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})\sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= P(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) \quad (\text{per la 14.5}) \end{aligned}$$

Di conseguenza la popolazione dei campioni dopo un ciclo di aggiornamento rappresenta correttamente il messaggio in avanti al tempo $t + 1$.

Il particle filtering è quindi *consistente*, ma è anche *efficiente*? Per molti casi pratici la risposta pare affermativa: quest'approccio sembra mantenere una buona approssimazione della vera distribuzione a posteriori usando un numero costante di campioni. La Figura 14.19 mostra che il particle filtering funziona bene sul problema di localizzazione in un mondo a griglia con solamente un migliaio di campioni. Il metodo funziona anche su problemi del mondo reale: l'algoritmo supporta migliaia di applicazioni in campo scientifico e ingegneristico (alcuni riferimenti sono forniti al termine del capitolo). Gestisce combinazioni di variabili discrete e continue, oltre a modelli non lineari e non gaussiani per variabili continue. Sotto certe ipotesi, in particolare che le probabilità nei modelli di transizione e sensoriale siano limitate e non possano assumere i valori 0 e 1, è anche possibile dimostrare che l'approssimazione mantiene con alta probabilità un errore limitato, come suggerisce la figura.

Anche l'algoritmo di particle filtering, comunque, ha dei punti deboli. Vediamo come opera per il mondo dell'aspirapolvere con l'aggiunta della sporcizia. Ricordiamo, dal Paragrafo 14.3.2, che l'aggiunta della sporcizia aumenta la dimensione dello spazio degli stati di un fattore 2^{42} , per cui l'inferenza HMM esatta diviene impraticabile. Vogliamo che il robot si muova a caso e costruisca una mappa delle posizioni in cui c'è sporcizia (questo è un semplice esempio di **localizzazione e mapping simultanei**, o SLAM, *simultaneous localization and mapping*, che tratteremo più in dettaglio nel Capitolo 26). Indichiamo con $Sporco_{i,t}$ che

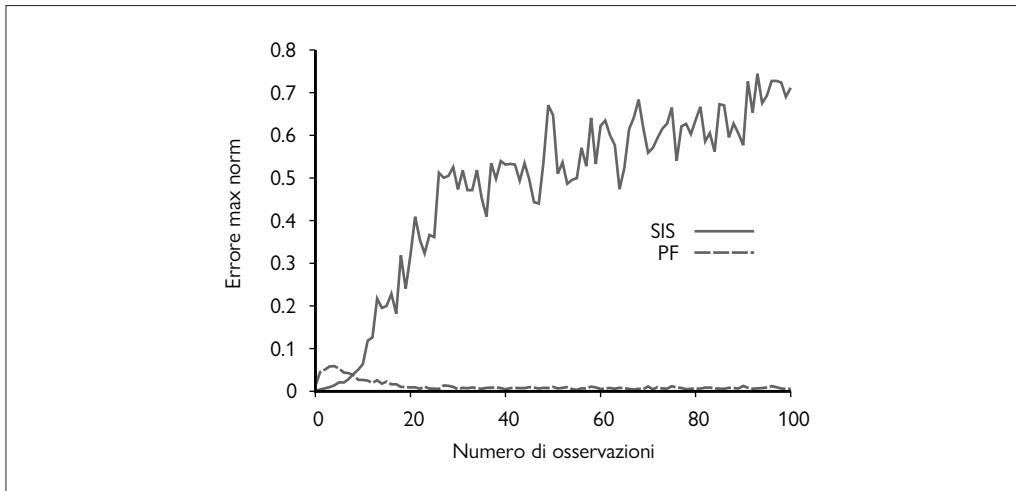


Figura 14.19 Errore max norm (distanza di Manhattan dalla vera posizione) nella stima della posizione sulla griglia (confrontato con l'inferenza esatta) per la pesatura di verosimiglianza (campionamento di importanza sequenziale) con 100.000 campioni e particle filtering con 1.000 campioni; media su 50 esecuzioni.

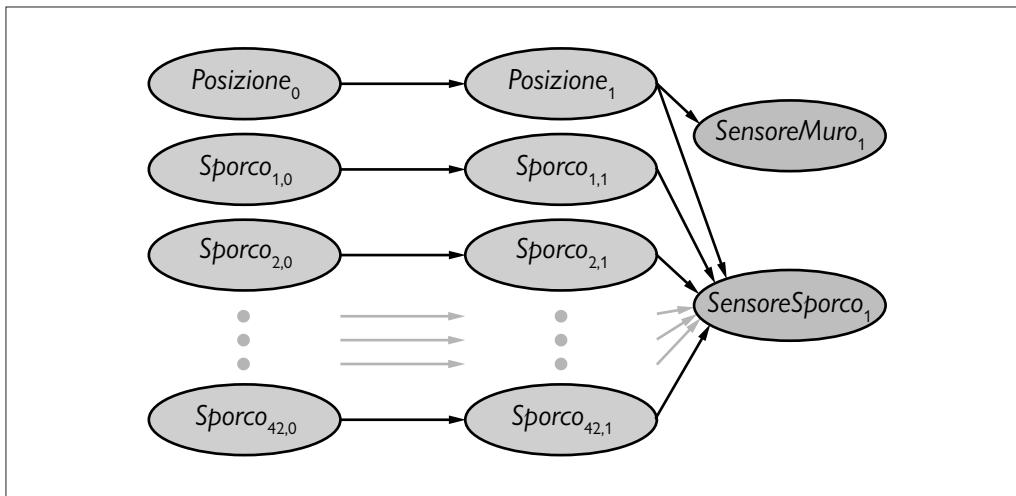


Figura 14.20 Una rete bayesiana dinamica per localizzazione e mapping simultanei nel mondo dell'aspirapolvere con sporcizia stocastica. I riquadri sporchi persistono con probabilità p , e quelli puliti diventano sporchi con probabilità $1 - p$. Il sensore di sporco locale è accurato al 90%, per il riquadro in cui si trova attualmente il robot.

il riquadro i è sporco al tempo t e sia $SensoreSporco_i$ vero se e solo se il robot rileva la presenza di sporcizia al tempo t . Ipotizzeremo che, in ogni riquadro dato, la sporcizia persista con probabilità p , mentre un riquadro pulito diventi sporco con probabilità $1 - p$ (il che significa che ogni riquadro è sporco per metà del tempo, in media). Il robot ha un sensore di sporcizia per la sua posizione corrente e tale sensore è accurato con probabilità 0,9. La Figura 14.20 mostra la DBN.

Per semplicità cominciamo ipotizzando che il robot disponga di un sensore di posizione perfetto, invece del sensore rumoroso per la presenza di muri. La prestazione dell'algoritmo è mostrata nella Figura 14.21(a), in cui si confrontano le sue stime di sporcizia con i risultati

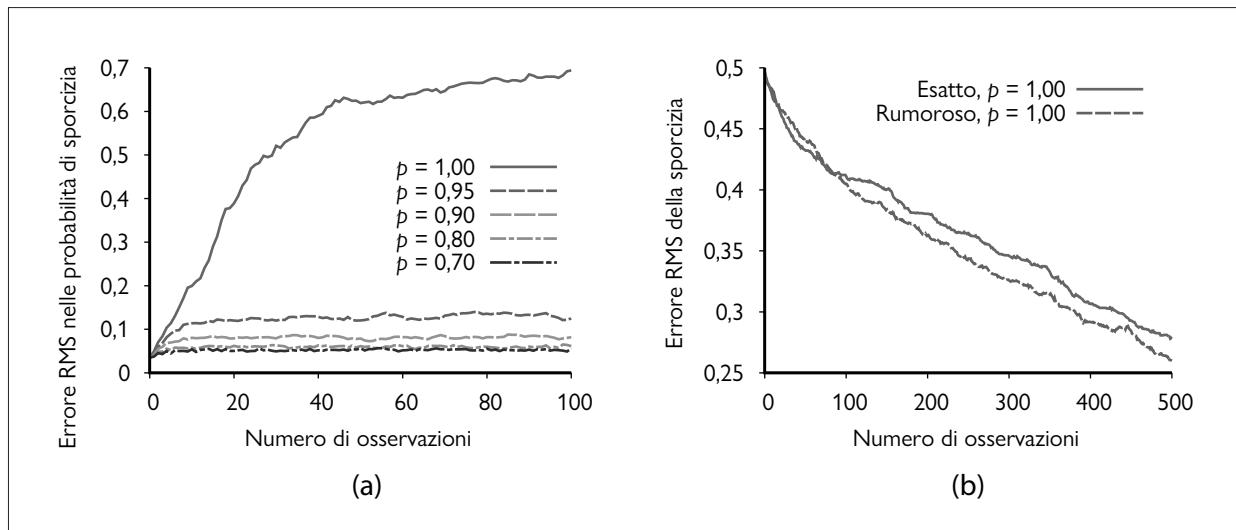


Figura 14.21 (a) Prestazione dell'algoritmo di particle filtering standard con 1.000 particelle, che mostra l'errore RMS (*root mean square*) nelle probabilità di sporcizia marginali rispetto all'inferenza esatta per valori diversi della persistenza dello sporco p . (b) Prestazione del particle filtering di Rao-Blackwell (100 particelle) a confronto con la vera presenza di sporcizia per il caso con percezione esatta della posizione e per quello con percezione rumorosa dei muri e, in entrambi i casi con sporco deterministico. Media su 20 esecuzioni.

dell'inferenza esatta (vedremo tra poco in che modo è possibile realizzare l'inferenza esatta). Per valori bassi della persistenza dello sporco p , l'errore rimane piccolo, ma questo non è un gran risultato, perché per ogni riquadro la vera distribuzione a posteriori dello sporco è vicina a quella che si otterrebbe se il robot non avesse visitato recentemente tale riquadro. Per valori più alti di p , la sporcizia persiste più a lungo, perciò la visita di un riquadro fornisce più informazioni utili che rimangono valide per un periodo più lungo. Può destare sorpresa il fatto che il particle filtering si comporti *peggior* per valori più elevati di p . Fallisce completamente quando $p = 1$, benché sembri il caso più facile: la sporcizia si presenta al tempo 0 e rimane per sempre, perciò dopo qualche perlustrazione del mondo il robot dovrebbe avere una mappa della sporcizia quasi perfetta. E allora perché il particle filtering fallisce in quel caso?

Risulta che la condizione teorica per cui “le probabilità nei modelli di transizione e sensoriale sono strettamente maggiori di 0 e minori di 1” non è una semplice pignoleria matematica. Per prima cosa, ogni particella contiene inizialmente 42 ipotesi da $\mathbf{P}(\mathbf{X}_0)$ su quali riquadri siano sporchi e quali no. Poi, lo stato per ogni particella viene proiettato in avanti nel tempo secondo il modello di transizione. Ma sfortunatamente il modello di transizione per la sporcizia deterministica è deterministico: la sporcizia rimane esattamente dove si trova. Quindi, le ipotesi iniziali in ogni particella non vengono mai aggiornate in base alle evidenze.

La possibilità che le ipotesi iniziali siano tutte corrette è 2^{-42} , circa 2×10^{-13} , perciò è del tutto improbabile che in un migliaio di particelle (o anche in un milione) ve ne sia una con la mappa dei riquadri sporchi corretta. Generalmente la migliore particella conterrà circa 32 riquadri giusti e 10 sbagliati, e solitamente ci sarà soltanto una particella come questa, o forse una manciata. Una di queste migliori particelle arriverà con il passare del tempo a dominare la probabilità totale e la diversità della popolazione di particelle crollerà. A quel punto, dato che tutte le particelle concorderanno su una singola mappa errata, l'algoritmo si convincerà che tale mappa è corretta e non cambierà più idea.

Fortunatamente il problema di localizzazione e mapping simultanei ha una struttura speciale: condizionatamente alla sequenza di posizioni del robot, gli stati di sporcizia dei singoli riquadri sono indipendenti (Esercizio 14.RBPF). Più specificamente,

$$\begin{aligned} \mathbf{P}(Sporco_{1,0:t}, \dots, Sporco_{42,0:t} | SensoreSporco_{1:t}, SensoreMuro_{1:t}, Posizione_{1:t}) \\ = \prod_i \mathbf{P}(Sporco_{i,0:t} | SensoreSporco_{1:t}, Posizione_{1:t}). \end{aligned} \quad (14.24)$$

Questo significa che è utile applicare un trucco statistico detto **tecnica di Rao-Blackwell**, basato sul concetto che l'inferenza esatta è sempre più accurata del campionamento, anche se soltanto per un sottoinsieme delle variabili (cfr. Esercizio 14.RAOB). Per il problema dello SLAM, eseguiamo il particle filtering sulla posizione del robot e poi, per ogni particella, effettuiamo l'inferenza HMM esatta per ogni riquadro sporco in modo indipendente, condizionatamente alla sequenza di posizioni in tale particella. Ogni particella, quindi, contiene una posizione campionata più 42 distribuzioni a posteriori marginali esatte per i 42 riquadri – esatte nel senso che si assume che la traiettoria ipotizzata per la particella sia corretta. Questo approccio, detto **particle filter di Rao-Blackwell**, gestisce senza difficoltà il caso della sporcizia deterministica, costruendo gradualmente una mappa esatta dei riquadri sporchi con la percezione esatta della posizione o la percezione rumorosa dei muri, come mostrato nella Figura 14.21(b).

**tecnica
di Rao-Blackwell**

Nei casi in cui non è soddisfatta la struttura di indipendenza condizionale esemplificata dall'Equazione (14.24), la tecnica di Rao-Blackwell non è applicabile. Nelle note storiche e bibliografiche al termine del capitolo sono citati numerosi algoritmi proposti per affrontare il problema generale del filtraggio con variabili statiche; nessuno presenta l'eleganza e l'ampia utilizzabilità del particle filtering, ma diversi sono efficaci nella pratica su alcune classi di problemi.

**particle filter
di Rao-Blackwell**

14.6 Riepilogo

Questo capitolo ha affrontato il problema generale della rappresentazione e del ragionamento sui processi temporali probabilistici. Questi sono i concetti fondamentali.

- La mutevolezza del mondo è gestita mediante un insieme di variabili casuali, che ne rappresentano lo stato in ogni istante.
- Le rappresentazioni possono essere progettate in modo da soddisfare (approssimativamente) la **proprietà di Markov**, così che il futuro possa essere indipendente dal passato dato il presente. Questa ipotesi, unita a quella che il processo sia **omogeneo rispetto al tempo**, permette di semplificare significativamente la rappresentazione.
- Si può pensare che un modello di probabilità temporale contenga un **modello di transizione** che descrive la sua evoluzione e un **modello sensoriale** che descrive il processo di osservazione.
- I principali compiti dell'inferenza nei modelli temporali sono il **filtraggio (stima dello stato)**, la **predizione**, lo **smoothing** o **regolarizzazione** e il calcolo della **spiegazione più probabile**. Ognuno di questi compiti può essere svolto facendo ricorso a semplici algoritmi ricorsivi che richiedono un tempo lineare nella lunghezza della sequenza.
- Tre famiglie di modelli temporali sono state studiate con maggior attenzione: i **modelli di Markov nascosti**, i **filtri di Kalman** e le **reti bayesiane dinamiche** (che includono i primi due come casi speciali).
- A meno che non vengano formulate ipotesi particolari, come nel caso dei filtri di Kalman, l'inferenza esatta con molte variabili di stato è computazionalmente intrattabile. Nella pratica, l'algoritmo di **particle filtering** e i suoi derivati costituiscono una famiglia efficace di algoritmi di approssimazione.

Note storiche e bibliografiche

Molte delle idee base per la stima dello stato dei sistemi dinamici sono dovute al matematico C. F. Gauss (1809), che formulò un algoritmo deterministico dei minimi quadrati per il problema della stima delle orbite partendo da osservazioni astronomiche. A. A. Markov (1913) sviluppò quella che fu poi chiamata **ipotesi di Markov** nella sua analisi dei processi stocastici; egli stimò una catena di Markov del primo ordine sulle lettere che compongono il testo dell'*Evgenij Onegin*. La teoria generale delle catene di Markov e dei loro tempi di mixing è trattata da Levin *et al.* (2008).

Durante la Seconda Guerra Mondiale furono svolte molte ricerche segrete sul filtraggio da Wiener (1942) per i processi a tempo continuo e da Kolmogorov (1941) per quelli a tempo discreto. Benché questo lavoro abbia portato a importanti sviluppi tecnologici nei vent'anni successivi, il suo utilizzo della rappresentazione nel dominio delle frequenze rendeva alcuni calcoli piuttosto difficoltosi. La modellazione diretta dei processi stocastici nello spazio degli stati si rivelò più semplice, come mostrarono Swerling (1959) e Kalman (1960). Quest'ultimo articolo introduceva quello che oggi è noto come filtro di Kalman per l'inferenza in avanti nei sistemi lineari con rumore gaussiano. I risultati di Kalman, comunque, erano già stati ottenuti in precedenza dall'astronomo danese Thorvold Thiele (1880) e dal fisico russo Ruslan Stratonovich (1959). Dopo una visita all'Ames Research Center della NASA nel 1960, Kalman intuì la possibilità di applicare il metodo al tracciamento delle traiettorie dei razzi, e il filtro fu poi implementato per le missioni Apollo.

Risultati fondamentali sullo smoothing furono derivati da Rauch *et al.* (1965), e il “regolarizzatore di Rauch–Tung–Striebel” è ancor oggi una tecnica standard. Molti dei primi risultati sono stati raccolti in Gelb (1974). Bar-Shalom e Fortmann (1988) offrono un trattamento moderno con un sapore bayesiano, insieme a molti riferimenti alla vasta letteratura sull'argomento. Chatfield (1989) e Box *et al.* (2016) trattano l'approccio della teoria del controllo all'analisi di serie temporali.

Il modello di Markov nascosto e i relativi algoritmi per l'inferenza e l'apprendimento, tra cui l'algoritmo forward–backward, fu sviluppato da Baum e Petrie (1966). L'algoritmo di Viterbi apparve per la prima volta in Viterbi (1967). Idee simili sono apparse, in modo indipendente, anche all'interno della comunità del filtraggio di Kalman (Rauch *et al.*, 1965).

L'algoritmo forward–backward è stato uno dei principali precursori della formulazione generale dell'algoritmo EM (Dempster *et al.*, 1977); ritorneremo sull'argomento nel Capitolo 20. Lo smoothing con requisiti spaziali costanti compare in Binder *et al.* (1997b), così come l'algoritmo *divide et impera* che sviluppiamo nell'Esercizio 14.ISLE. Lo smoothing a ritardo fisso eseguito in tempo costante per HMM fu presentato per la prima volta in Russell e Norvig (2003).

I modelli HMM hanno trovato molte applicazioni nell'elaborazione del linguaggio (Charniak, 1993), nel riconoscimento vocale (Rabiner e Juang, 1993), nella traduzione automatica (Och e Ney, 2003), nella biologia computazionale (Krogh *et al.*, 1994; Baldi *et al.*, 1994), nell'economia finanziaria (Bhar e Hamori, 2004) e in altri campi. Il modello HMM è stato esteso in vari modi: per esempio, i modelli HMM gerarchici (Fine *et al.*, 1998) e HMM stratificati (Oliver *et al.*, 2004) introducono una struttura nel modello, sostituendo la singola variabile di stato dei modelli HMM standard.

Le reti bayesiane dinamiche (DBN) possono essere considerate una codifica sparsa di un processo di Markov e sono state usate per la prima volta nell'IA da Dean e Kanazawa (1989b), Nicholson (1992) e Kjaerulff (1992). Quest'ultimo lavoro include un'estensione del sistema HUGIN che impiega reti bayesiane dinamiche. Il libro di Dean e Wellman (1991) ha favorito la diffusione delle DBN e dell'approccio probabilistico alla pianificazione e al controllo nell'ambito dell'IA. Murphy (2002) presenta un'analisi completa delle DBN.

Le DBN sono diventate popolari anche per la modellazione di processi di moto complessi nel campo della visione artificiale (Huang *et al.*, 1994; Intille e Bobick, 1999). Come i modelli HMM, le DBN hanno trovato applicazioni in campi come riconoscimento vocale (Zweig e Russell, 1998; Livescu *et al.*, 2003), localizzazione di robot (Theocharous *et al.*, 2004), genetica (Murphy e Mian, 1999; Li *et al.*, 2011). Tra gli altri campi di applicazione vi sono l'analisi dei gesti (Suk *et al.*, 2010), il rilevamento dell'affaticamento del guidatore (Yang *et al.*, 2010) e la modellizzazione del traffico urbano (Hofleitner *et al.*, 2012).

Il collegamento tra HMM e DBN, e tra l'algoritmo forward–backward e la propagazione nelle reti bayesiane, fu reso esplicito da Smyth *et al.* (1997). Un'ulteriore unificazione con i filtri di Kalman (e altri mo-

delli statistici) compare in Roweis e Ghahramani (1999). Esistono procedure per l'apprendimento di parametri (Binder *et al.*, 1997a; Ghahramani, 1998) e strutture (Friedman *et al.*, 1998) delle DBN. Le **reti bayesiane in tempo continuo** (Nodelman *et al.*, 2002) sono l'analogo delle DBN a stato discreto e tempo continuo, in cui si evita le necessità di scegliere una particolare durata per i passi temporali.

I primi algoritmi di campionamento per il filtraggio (chiamati anche metodi Monte Carlo sequenziali) furono sviluppati nella comunità della teoria del controllo da Handschin e Mayne (1969), e l'idea del ricampionamento che è alla base del particle filtering comparve in una rivista russa dedicata al controllo (Zaritskii *et al.*, 1975). Più tardi fu reinventata all'interno della statistica con il nome di **campionamento di importanza sequenziale con ricampionamento o SIR** (Rubin, 1988; Liu e Chen, 1998), nella teoria del controllo come particle filtering (Gordon *et al.*, 1993; Gordon, 1994), nell'IA come **sopravvivenza del più adatto** (Kanazawa *et al.*, 1995) e nella visione artificiale come **condensazione** (Isard e Blake, 1996).

L'articolo di Kanazawa *et al.* (1995) include una miglioria chiamata **inversione delle evidenze** in cui lo stato al tempo $t + 1$ è campionato condizionandolo sia allo stato al tempo t che all'evidenza al tempo $t + 1$. Questo permette alle evidenze di influenzare direttamente la generazione dei campioni; Doucet (1997) e Liu e Chen (1998) hanno dimostrato che ciò riduce l'errore dell'approssimazione.

Il particle filtering ha trovato applicazione in molti campi, tra cui il tracciamento di schemi di movimento complessi nei video (Isard e Blake, 1996), la predizione dei mercati azionari (de Freitas *et al.*, 2000) e la diagnostica degli errori nei rover di esplorazione dei pianeti (Verma *et al.*, 2004). Da quando l'algoritmo è stato inventato, sono state pubblicate decine di migliaia di articoli su applicazioni e varianti. Le implementazioni scalabili su hardware parallelo sono divenute importanti; benché possa sembrare semplice distribuire N particelle su un massimo di N thread di un processore, l'algoritmo di base richiede la comunicazione sincronizzata fra i thread per il ricampionamento (Hendey *et al.*, 2010). Nell'**algoritmo a cascata di particelle** (Paige *et al.*, 2015) il requisito della sincronizzazione non c'è, per cui il calcolo parallelo è molto più veloce.

Il **particle filter di Rao-Blackwell** si deve a Doucet *et al.* (2000) e Murphy e Russell (2001); la sua appli-

cazione a problemi concreti di localizzazione e mapping in robotica è descritta nel Capitolo 26 del Volume 2. Sono stati proposti molti altri algoritmi per gestire problemi di filtraggio più generali con variabili statiche o quasi statiche, tra cui l'algoritmo ricampiona-muovi (Gilks e Berzuini, 2001), l'algoritmo di Liu-West (Liu e West, 2001), il filtro di Storvik (Storvik, 2002), l'extended parameter filter (Erol *et al.*, 2013), e l'assumed parameter filter (Erol *et al.*, 2017). Quest'ultimo è un ibrido tra particle filtering e un'idea molto più antica: l'**assumed-density filter**. Un assumed-density filter assume che la distribuzione a posteriori sugli stati al tempo t appartenga a una particolare famiglia a parametrizzazione finita; se i passi di proiezione e aggiornamento la portano al di fuori di tale famiglia, la distribuzione viene proiettata all'indietro per fornire la migliore approssimazione all'interno della famiglia. Per le DBN, l'algoritmo di Boyen-Koller (Boyen *et al.*, 1999) e l'algoritmo della **frontiera fattorizzata** (Murphy e Weiss, 2001) assumono che la distribuzione a posteriori possa essere approssimata bene da un prodotto di piccoli fattori.

I metodi MCMC (cfr. Paragrafo 13.4.2) possono essere applicati al problema del filtraggio; per esempio, il campionamento di Gibbs può essere applicato direttamente a una DBN non srotolata. La famiglia di algoritmi **particle MCMC** (Andrieu *et al.*, 2010; Lindsten *et al.*, 2014) combina metodi MCMC sul modello temporale non srotolato e particle filtering per generare le proposte MCMC; benché si possa dimostrare che converge alla distribuzione a posteriori corretta nel caso generale (cioè con variabili sia statiche sia dinamiche), è un algoritmo offline. Per evitare il problema dei tempi di aggiornamento crescenti al crescere della rete non srotolata, il filtro **decayed MCMC** (Marthi *et al.*, 2002) preferisce campionare variabili di stato più recenti, con una probabilità che decresce per variabili più lontane nel passato.

Il libro di Doucet *et al.* (2001) raccoglie molti importanti articoli sugli algoritmi **Monte Carlo sequenziali** (SMC), di cui il particle filtering costituisce l'esempio più importante. Sono disponibili anche utili tutorial di Arulampalam *et al.* (2002) e Doucet e Johansen (2011). Esistono anche diversi risultati teorici relativi alle condizioni sotto le quali i metodi SMC mantengono sempre un errore limitato rispetto alla vera distribuzione a posteriori (Crisan e Doucet, 2002; Del Moral, 2004; Del Moral *et al.*, 2006).

Programmazione probabilistica

- 15.1 Modelli relazionali di probabilità
- 15.2 Modelli probabilistici a universo aperto
- 15.3 Tenere traccia di un mondo complesso
- 15.4 Programmi come modelli probabilistici
- 15.5 Riepilogo
Note storiche e bibliografiche

In cui spieghiamo il concetto di linguaggi universali per la rappresentazione della conoscenza probabilistica e per l'inferenza in domini incerti.

La gamma delle rappresentazioni, atomiche, fattorizzate e strutturate, è un tema persistente nell'ambito dell'IA. Per i modelli deterministici, gli algoritmi di ricerca presuppongono solo una rappresentazione atomica; i CSP e la logica proposizionale forniscono rappresentazioni fattorizzate; e la logica del primo ordine e i sistemi di pianificazione sfruttano le rappresentazioni strutturate. La capacità espressiva concessa dalle rappresentazioni strutturate produce modelli molto più concisi delle equivalenti descrizioni fattorizzate o atomiche.

Per i modelli probabilistici, le reti bayesiane descritte nei Capitoli 13 e 14 sono rappresentazioni fattorizzate: l'insieme di variabili casuali è fisso e finito, e ogni variabile ha un intervallo fisso di valori possibili. Questo limita l'applicabilità delle reti bayesiane, perché la rete bayesiana che rappresenta un dominio complesso è semplicemente troppo grande. Ciò rende impossibile costruire manualmente queste rappresentazioni e altrettanto impossibile apprenderle da una quantità ragionevole di dati.

Il problema di creare un linguaggio formale espressivo per le informazioni probabilistiche ha messo alla prova alcune delle più grandi menti della storia, tra cui Gottfried Leibniz (co-inventore del calcolo infinitesimale), Jacob Bernoulli (scopritore di e , del calcolo delle variazioni e della legge dei grandi numeri), Augustus De Morgan, George Boole, Charles Sanders Peirce (uno dei maggiori logici del XIX secolo), John Maynard Keynes (l'economista più importante del XX secolo) e Rudolf Carnap (uno dei più grandi filosofi analitici del XX secolo). Il problema ha resistito a questi e ad altri tentativi fino agli anni 1990.

Anche grazie allo sviluppo delle reti bayesiane, oggi esistono linguaggi formali matematicamente eleganti e altamente funzionali che consentono di creare modelli probabilistici per domini molto complessi. Questi linguaggi sono *universal* nello stesso senso in cui lo sono le macchine di Turing: possono rappresentare qualsiasi modello di probabilità calcolabile, come le macchine di Turing possono rappresentare qualsiasi funzione calcolabile. Inoltre, questi linguaggi sono accompagnati da algoritmi di inferenza generali, più o meno analoghi agli algoritmi di inferenza logica corretti e completi, come la risoluzione.

Esistono due strade per introdurre capacità espressiva all'interno della teoria della probabilità. La prima è la via logica: individuare un linguaggio che definisca le probabilità sui mondi possibili del primo ordine, invece che sui mondi possibili proposizionali delle reti bayesiane. Di questa via si occupano i Paragrafi 15.1 e 15.2, mentre il Paragrafo 15.3 affronta il caso specifico del ragionamento temporale. La seconda via è quella dei linguaggi di programmazione tradizionali: si introducono elementi stocastici (scelte casuali, per esempio) in questi linguaggi e si considera che i programmi definiscano distribuzioni di probabilità sui propri tracciati di esecuzione. Questo approccio è descritto nel Paragrafo 15.4.

linguaggio di programmazione probabilistico (PPL)

Entrambe le vie conducono a un **linguaggio di programmazione probabilistico (PPL)**, da *probabilistic programming language*). La prima conduce a PPL di tipo dichiarativo, che hanno con i PPL generali la stessa relazione che la programmazione logica (Capitolo 9) ha con i linguaggi di programmazione generali.

15.1 Modelli relazionali di probabilità

Come abbiamo visto nel Capitolo 12, un modello probabilistico definisce un insieme Ω di mondi possibili, dove ogni mondo ω ha probabilità $P(\omega)$. Per le reti bayesiane, i mondi possibili sono assegnamenti di valori a delle variabili; in particolare, per il caso booleano i mondi possibili sono identici a quelli della logica proposizionale.

Per un modello probabilistico del primo ordine, quindi, sembra necessario che i mondi possibili siano quelli della logica del primo ordine; ovvero, un insieme di oggetti legati da relazioni e un'interpretazione che associa simboli di costante a oggetti, simboli di predicato a relazioni e simboli di funzione a funzioni su tali oggetti (cfr. Paragrafo 8.2). Il modello deve anche definire una probabilità per ognuno di questi mondi possibili, così come una rete bayesiana definisce una probabilità per ogni assegnamento di valori alle variabili.

Supponiamo per un momento di avere capito come fare tutto ciò. Allora, come di consueto (cfr. Paragrafo 12.2.1), possiamo ricavare la probabilità di ogni formula logica del primo ordine ϕ (phi) come sommatoria sui mondi possibili nei quali è vera:

$$P(\phi) = \sum_{\omega: \phi \text{ è vera in } \omega} P(\omega). \quad (15.1)$$

Le probabilità condizionate $P(\phi | \mathbf{e})$ si possono ricavare in modo simile; quindi, in linea di principio, possiamo porre al modello qualsiasi domanda e ottenere una risposta. Fin qui, tutto bene.

C'è però un problema: l'insieme dei modelli del primo ordine è infinito. Lo si vede chiaramente nella Figura 8.4 del Capitolo 8, che riproponiamo qui nella Figura 15.1 (in alto). Ciò significa che (1) la sommatoria dell'Equazione (15.1) potrebbe essere non calcolabile e (2) specificare una distribuzione completa e coerente su un insieme infinito di mondi potrebbe essere molto difficile.

In questo paragrafo evitiamo questo problema considerando la **semantica dei database** definita nel Paragrafo 8.2.8 del Capitolo 8. La semantica dei database comprende l'**ipotesi dei nomi unici**, che in questo contesto adottiamo per i simboli di costante. Presuppone anche la **chiusura del dominio**, ovvero che non ci siano altri oggetti oltre a quelli denominati. Pos-

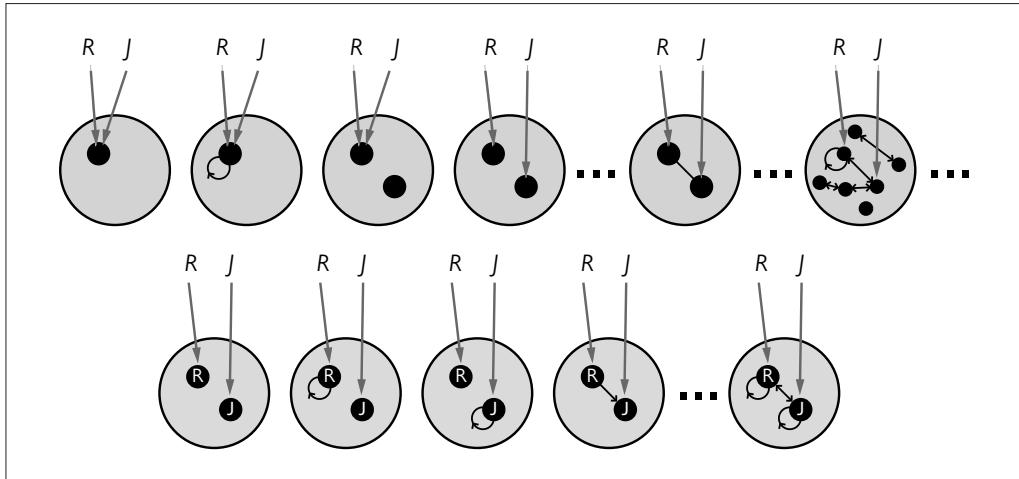


Figura 15.1 In alto: alcuni elementi dell'insieme di tutti i mondi possibili per un linguaggio con due simboli di costante, R e J , e un simbolo di relazione binaria, secondo la semantica standard della logica del primo ordine. In basso: i mondi possibili secondo la semantica dei database. L'interpretazione dei simboli di costante è fissata e c'è un oggetto distinto per ogni simbolo di costante.

siamo quindi garantire che l'insieme dei mondi possibili sia finito stabilendo che in ogni mondo l'insieme degli oggetti sia esattamente l'insieme dei simboli di costante che vengono utilizzati; come mostrato nella Figura 15.1 (in basso), non c'è incertezza sull'associazione dei simboli agli oggetti o su quali oggetti esistano.

Chiameremo i modelli definiti in questo modo **modelli relazionali di probabilità**, o RPM (*relational probability models*).¹ La differenza più significativa tra la semantica dei RPM e quella dei database presentata nel Paragrafo 8.2.8 è che i RPM non fanno l'ipotesi del mondo chiuso; in un sistema di ragionamento probabilistico non è possibile assumere che ogni fatto sconosciuto sia falso.

modello relazionale
di probabilità

15.1.1 Sintassi e semantica

Iniziamo con un semplice esempio: supponiamo che un negozio di libri online voglia offrire valutazioni dei prodotti basate sui suggerimenti ricevuti dai clienti. La valutazione assumerà la forma di una distribuzione a posteriori sulla qualità del libro, date le evidenze disponibili. La soluzione più semplice è basare la valutazione sul suggerimento medio, eventualmente con una varianza determinata dal numero di suggerimenti, ma così facendo non si prende in considerazione il fatto che alcuni clienti sono più gentili di altri e alcuni sono meno onesti di altri. I clienti gentili tendono a dare giudizi alti anche a libri piuttosto normali, mentre quelli disonesti danno giudizi molto alti o molto bassi per ragioni diverse dalla qualità; potrebbero per esempio essere pagati per promuovere i libri di un certo editore.²

Per un singolo cliente C_1 che suggerisce un singolo libro L_1 , la rete bayesiana potrebbe essere simile a quella della Figura 15.2(a) (come nel Paragrafo 9.1, le espressioni con parentesi come $Onestà(C_1)$ sono solo simboli; in questo caso nomi di variabili casuali). Con due clienti e due libri, la rete bayesiana è simile a quella della Figura 15.2(b). Per quantità maggiori di libri e di clienti, non è molto pratico specificare manualmente una rete bayesiana.

¹ Il nome *modello relazionale di probabilità* è stato assegnato da Pfeffer (2000) a una rappresentazione leggermente diversa, ma le idee di base sono le stesse.

² Un esperto di teoria dei giochi consiglierebbe a un cliente disonesto di suggerire occasionalmente un buon libro di un concorrente, per evitare di farsi scoprire. Cfr. il Capitolo 18.

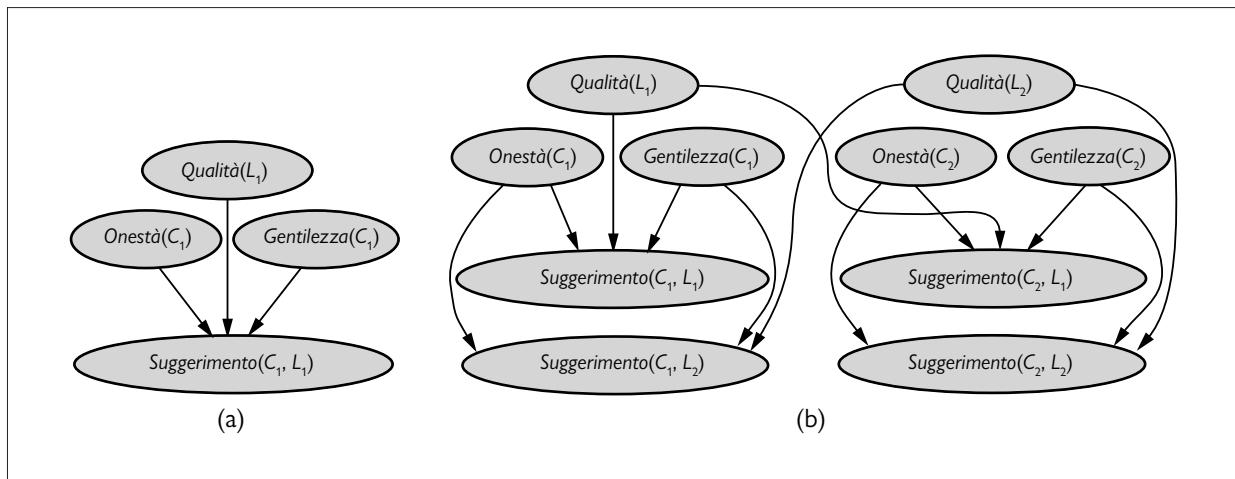


Figura 15.2 (a) Rete bayesiana per un singolo cliente C_1 che suggerisce un singolo libro L_1 . $Onestà(C_1)$ è booleana, mentre le altre variabili hanno valori interi compresi tra 1 e 5. (b) Rete bayesiana con due clienti e due libri.

Fortunatamente, nella rete molte strutture si ripetono. Ogni variabile $Suggerimento(c, l)$ ha come genitori le variabili $Onestà(c)$, $Gentilezza(c)$ e $Qualità(l)$. Inoltre, le tabelle delle probabilità condizionate (CPT) sono identiche per tutte le variabili $Suggerimento(c, l)$, così come quelle per le variabili $Onestà(c)$ e così via. La situazione sembra perfetta per un linguaggio del primo ordine. Verrebbe spontaneo affermare qualcosa come:

$$Suggerimento(c, l) \sim CPT Sugg(Onestà(c), Gentilezza(c), Qualità(l))$$

che significa che il suggerimento di un cliente riguardo a un libro dipende probabilisticamente dall'onestà e dalla gentilezza del cliente e dalla qualità del libro, in base a una CPT stabilita.

Come la logica del primo ordine, i modelli RPM hanno simboli di costante, di funzione e di predicato. Assumeremo anche una **dichiarazione dei tipi** per ogni funzione, ovvero una specifica del tipo per ciascun argomento e per il valore della funzione (se è noto il tipo di ogni oggetto, molti possibili mondi spuri vengono eliminati da questo meccanismo; per esempio, non occorre occuparsi della gentilezza dei libri, di libri che suggeriscono clienti e così via). Nel dominio dei suggerimenti sui libri, i tipi sono *Cliente* e *Libro*, e le dichiarazioni dei tipi delle funzioni e dei prediciati sono le seguenti:

$$\begin{aligned} Onestà : & Cliente \rightarrow \{\text{true}, \text{false}\} \\ Gentilezza : & Cliente \rightarrow \{1, 2, 3, 4, 5\} \\ Qualità : & Libro \rightarrow \{1, 2, 3, 4, 5\} \\ Suggerimento : & Cliente \times Libro \rightarrow \{1, 2, 3, 4, 5\}. \end{aligned}$$

I simboli di costante saranno i nomi dei clienti e dei libri che compaiono nei dati del venditore; nell'esempio della Figura 15.2(b) sono C_1 , C_2 e L_1 , L_2 .

dichiarazione dei tipi

variabile casuale di base

Date le costanti e i loro tipi, le funzioni e le loro dichiarazioni dei tipi, le **variabili casuali di base** del modello RPM si ottengono creando un'istanza di ogni funzione con ogni possibile combinazione di oggetti. Per il modello dei suggerimenti sui libri, le variabili casuali base sono $Onestà(C_1)$, $Qualità(L_2)$, $Suggerimento(C_1, L_2)$ e così via. Sono esattamente le variabili che appaiono nella Figura 15.2(b). Poiché ciascun tipo ha solamente un numero finito di istanze (grazie all'assunzione della chiusura del dominio), è finito anche il numero delle variabili casuali di base.

Per completare il modello RPM, dobbiamo scrivere le dipendenze che governano queste variabili casuali. Per ogni funzione c'è una dichiarazione delle dipendenze, in cui ogni argomento della funzione è una variabile logica (cioè una variabile definita sugli oggetti, come nella logica del primo ordine). Per esempio, la dichiarazione seguente afferma che, per ogni cliente c la probabilità a priori dell'onestà è 0,99 *true* e 0,01 *false*:

$$\text{Onestà}(c) \sim \langle 0,99, 0,01 \rangle.$$

Analogamente, possiamo dichiarare le probabilità a priori per il valore di gentilezza di ogni cliente e per la qualità di ogni libro, in entrambi i casi su una scala 1–5:

$$\text{Gentilezza}(c) \sim \langle 0,1, 0,1, 0,2, 0,3, 0,3 \rangle$$

$$\text{Qualità}(l) \sim \langle 0,05, 0,2, 0,4, 0,2, 0,15 \rangle.$$

Infine, occorrono le dipendenze per i suggerimenti: per ogni cliente c e libro l , il punteggio dipende dall'onestà e dalla gentilezza del cliente e dalla qualità del libro:

$$\text{Suggerimento}(c, l) \sim CPT\text{Sugg}(\text{Onestà}(c), \text{Gentilezza}(c), \text{Qualità}(l))$$

dove $CPT\text{Sugg}$ è una tabella di probabilità condizionata definita separatamente con $2 \times 5 \times 5 = 50$ righe con 5 elementi ciascuna. A scopo illustrativo, assumiamo che un suggerimento onesto per un libro di qualità q formulato da una persona di gentilezza g sia uniformemente distribuito sull'intervallo $\left[\left\lfloor \frac{q+k}{2} \right\rfloor, \left\lceil \frac{q+k}{2} \right\rceil\right]$.

La semantica del modello RPM può essere ricavata creando istanze di queste dipendenze per tutte le costanti note, il che conduce a una rete bayesiana (come nella Figura 15.2(b)) che definisce una distribuzione congiunta sulle variabili casuali del modello RPM.³

L'insieme dei mondi possibili è il prodotto cartesiano degli intervalli su cui sono definite tutte le variabili casuali di base e, come nelle reti bayesiane, la probabilità di ogni mondo possibile è il prodotto delle probabilità condizionate rilevanti del modello. Con C clienti e L libri, ci sono C variabili *Onestà*, C variabili *Gentilezza*, L variabili *Qualità* e LC variabili *Suggerimento*, che risultano in $2^C 5^{C+L+LC}$ possibili mondi. Con dieci milioni di libri e un miliardo di clienti, sarebbero circa $10^{7 \times 10^{15}}$ mondi. Grazie alla capacità espressiva dei RPM, il modello probabilistico completo ha comunque meno di 300 parametri, la maggior parte dei quali nella tabella *CPTsugg*.

Possiamo perfezionare il modello affermando una **indipendenza specifica del contesto** (cfr. Paragrafo 13.2.2) per riflettere il fatto che i clienti disonesti non considerano la qualità, nel dare suggerimenti; anche la gentilezza non ha ruolo nelle loro decisioni. Quindi, *Suggerimento*(c, l) è indipendente da *Gentilezza*(c) e da *Qualità*(l) quando *Onestà*(c) = *false*:

$$\begin{aligned} \text{Suggerimento}(c, l) \sim & \quad \text{if } \text{Onestà}(c) \text{ then} \\ & \quad CPT\text{SuggOnesto}(\text{Gentilezza}(c), \text{Qualità}(l)) \\ & \quad \text{else } \langle 0,4, 0,1, 0,0, 0,1, 0,4 \rangle. \end{aligned}$$

Questo tipo di dipendenza può sembrare una normale istruzione if–then–else di un linguaggio di programmazione, ma c'è una differenza fondamentale: il motore di inferenza *non necessariamente conosce il valore del test condizionale* perché *Onestà*(c) è una variabile casuale.

Possiamo elaborare questo modello in infiniti modi per renderlo più realistico. Per esempio, supponiamo che un cliente onesto, ammiratore dell'autore di un libro, dia sempre il voto 5 al libro indipendentemente dalla qualità:

³ Affinché un modello RPM definisca una distribuzione appropriata sono necessarie alcune condizioni tecniche. Primo, le dipendenze devono essere *acicliche*; diversamente la rete bayesiana risultante avrà dei cicli. Secondo, le dipendenze devono (solitamente) essere *ben fondate*: non possono esserci catene di ascendenti infinite, come potrebbe accadere con le dipendenze ricorsive. Un'eccezione a questa regola si trova nell'Esercizio 15.HAMD.

```

Suggerimento( $c, l$ ) ~ if Onestà( $c$ ) then
    if Ammiratore( $c, Autore(l)$ ) then Esattamente(5)
    else CPTSGuggOnesto(Gentilezza( $c$ ), Qualità( $l$ ))
else ⟨0,4, 0,1, 0,0, 0,1, 0,4⟩

```

Anche qui il test condizionale $Ammiratore(c, Autore(l))$ è sconosciuto, ma se un cliente assegna sempre il voto 5 ai libri di un particolare autore e se in generale non è particolarmente gentile, allora la probabilità a posteriori che il cliente sia un ammiratore di quell'autore è alta. Inoltre, la distribuzione a posteriori tenderà a sottopesare i 5 attribuiti dal cliente, nella valutazione della qualità dei libri dell'autore in questione.

In questo esempio abbiamo implicitamente assunto che il valore di $Autore(l)$ fosse noto per ogni l , ma potrebbe non essere così. In che modo il sistema può tenere conto del fatto che C_1 , per esempio, sia un ammiratore di $Autore(L_2)$, quando $Autore(L_2)$ è sconosciuto? La risposta è che potrebbe essere necessario che il sistema tenga conto di *tutti gli autori possibili*. Supponiamo (per semplicità) che ci siano solamente due autori, A_1 e A_2 . Allora $Autore(L_2)$ è una variabile casuale con due possibili valori, A_1 e A_2 , ed è genitore di $Suggerimento(C_1, L_2)$. Anche le variabili $Ammiratore(C_1, A_1)$ e $Ammiratore(C_1, A_2)$ sono genitori. La distribuzione condizionata di $Suggerimento(C_1, L_2)$ è allora essenzialmente un **multiplexer** in cui il genitore $Autore(L_2)$ agisce come selettore per scegliere quale tra $Ammiratore(C_1, A_1)$ e $Ammiratore(C_1, A_2)$ influenzi effettivamente il suggerimento. Una porzione dell'equivalente rete bayesiana è mostrata nella Figura 15.3. L'incertezza sul valore di $Autore(L_2)$, che influenza sulla struttura delle dipendenze della rete, è un caso di **incertezza relazionale**.

Se vi state domandando in che modo il sistema possa scoprire chi sia l'autore di L_2 , considerate la possibilità che altri tre clienti siano ammiratori di A_1 (e non abbiano altri autori preferiti in comune) e che tutti e tre abbiano dato a L_2 il voto 5, sebbene la maggior parte degli altri clienti lo trovi abbastanza scadente. In tal caso, è estremamente probabile che A_1 sia l'autore di L_2 . L'emergere di ragionamenti sofisticati come questo da un modello RPM di poche righe è un esempio affascinante di come le influenze probabilistiche si propaghino attraverso la rete di interconnessioni tra gli oggetti del modello. Man mano che si aggiungono ulteriori dipendenze e oggetti, spesso accade che il quadro su cui la distribuzione a posteriori è basata diventi via via più chiaro.

15.1.2 Esempio: valutare il livello di abilità dei giocatori

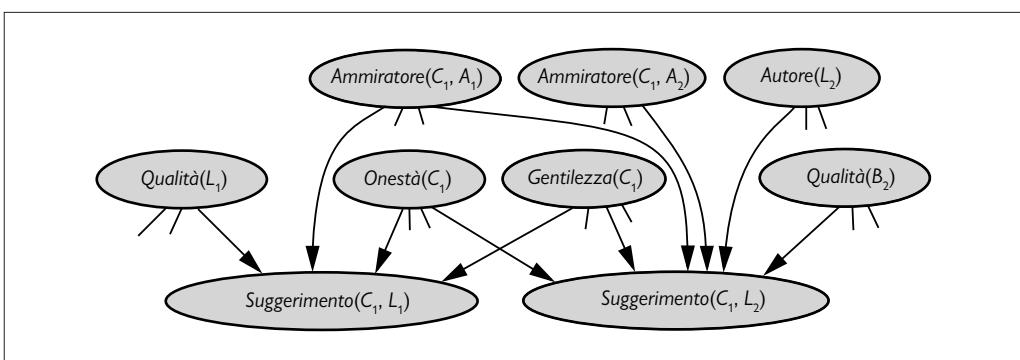
In molti giochi competitivi esiste una misura numerica del livello di abilità dei giocatori, chiamato a volte **rating** o punteggio. Il più noto è forse il punteggio Elo degli scacchisti, che tipicamente è attorno a 800 per un principiante e al di sopra di 2800 per il campione del mondo. Sebbene abbiano una base statistica, i punteggi Elo hanno anche alcuni elementi ad hoc.

multiplexer

incertezza
relazionale

rating

Figura 15.3
Porzione della rete bayesiana equivalente per il modello RPM dei suggerimenti sui libri quando $Autore(L_2)$ è sconosciuto.



È possibile sviluppare uno schema di rating bayesiano nel modo seguente: ogni giocatore i ha un determinato livello di abilità $Abilità(i)$; la prestazione effettiva di i in una partita p è $Prestazione(i, p)$, che può differire rispetto al livello di abilità; e il vincitore di p è il giocatore la cui prestazione in p è la migliore. Come un RPM, il modello appare così:

$$Abilità(i) \sim N(\mu, \sigma^2)$$

$$Prestazione(i, g) \sim N(Abilità(i), \beta^2)$$

$$Vince(i, j, g) = \text{if } Gioco(g, i, j) \text{ then } (Prestazione(i, g) > Prestazione(j, g))$$

dove β^2 è la varianza della prestazione effettiva di un giocatore in una specifica partita rispetto al livello di abilità base del giocatore stesso. Dato un insieme di giocatori, di partite e di risultati di alcune partite, un motore di inferenza RPM può calcolare una distribuzione a posteriori sull'abilità di ogni giocatore e il probabile esito di qualsiasi ulteriore partita.

Per i giochi di squadra assumiamo, come prima approssimazione, che la prestazione complessiva della squadra s sia la somma delle prestazioni individuali dei giocatori di s :

$$PrestazioneSquadra(s, p) = \sum_{i \in s} Prestazione(i, p).$$

Sebbene le prestazioni individuali non siano visibili al sistema di rating, i livelli di abilità dei giocatori possono comunque essere stimati partendo dai risultati di una serie di partite, a patto che la composizione delle squadre cambi tra una partita e l'altra. Il motore di rating Microsoft TrueSkill™ utilizza questo modello, assieme a un efficiente algoritmo di inferenza approssimata, per fornire servizi a milioni di utenti al giorno.

Questo modello può essere arricchito in numerosi modi. Per esempio, potremmo assumere che i giocatori più deboli abbiano prestazioni con varianza più alta, considerare il ruolo che il giocatore ha nella squadra e distinguere specifici tipi di prestazione e di abilità (per esempio difesa e attacco), per migliorare la composizione della squadra e la precisione predittiva.

15.1.3 Inferenza nei modelli relazionali di probabilità

L'approccio più semplice all'inferenza nei modelli RPM consiste semplicemente nel costruire la rete bayesiana equivalente, dati i simboli di costante noti appartenenti a ciascun tipo. Con L libri e C clienti, l'elementare modello descritto in precedenza potrebbe essere costruito con semplici iterazioni:⁴

```

for  $l = 1$  to  $L$  do
    aggiungi nodo  $Qualità_l$  senza genitori, con probabilità a priori  $\langle 0,05, 0,2, 0,4, 0,2, 0,15 \rangle$ 
for  $c = 1$  to  $C$  do
    aggiungi nodo  $Onestà_c$  senza genitori, con probabilità a priori  $\langle 0,99, 0,01 \rangle$ 
    aggiungi nodo  $Gentilezza_c$  senza genitori, con probabilità a priori  $\langle 0,1, 0,1, 0,2, 0,3, 0,3 \rangle$ 
    for  $l = 1$  to  $L$  do
        aggiungi nodo  $Suggerimento_{c,l}$  con genitori  $Onestà_c$ ,  $Gentilezza_c$ ,  $Qualità_l$ 
        e distribuzione condizionata  $CPTsugg(Onestà_c, Gentilezza_c, Qualità_l)$ .
```

⁴ Diversi programmi statistici considererebbero questo codice come ciò che *definisce* il modello RPM, e non solo come il costruttore di una rete bayesiana per compiere inferenze nel modello RPM. Così, tuttavia, mancherebbero di riconoscere un ruolo importante della sintassi del modello RPM: in mancanza di una sintassi dalla semantica chiara, è impossibile che la struttura del modello possa essere appresa dai dati.

**grounding
unrolling**

Questa tecnica è chiamata **grounding** (radicamento) o **unrolling** (srotolamento); è l'esatto analogo della **proposizionalizzazione** per la logica del primo ordine (Paragrafo 9.1). L'ovvio inconveniente è che la rete bayesiana risultante potrebbe essere molto grande. Inoltre, se per una relazione o funzione sconosciuta (per esempio, lo sconosciuto autore di L_2) i possibili oggetti candidati sono molti, allora alcune variabili della rete potrebbero avere molti genitori.

Fortunatamente, spesso è possibile evitare di generare l'intera rete bayesiana implicita. Come abbiamo visto nella discussione dell'algoritmo di eliminazione di variabili del Paragrafo 13.3.2, ogni variabile che non è ascendente di una variabile di query o di una variabile di evidenza è irrilevante per la query. Inoltre, se la query è condizionalmente indipendente rispetto a qualche variabile date le evidenze, allora è irrilevante anche tale variabile. Quindi, percorrendo il modello partendo dalla query e dalle evidenze, possiamo individuare l'insieme delle variabili che sono rilevanti per la query. Sono le sole che debbano essere istanziate per creare una porzione potenzialmente minuscola della rete bayesiana implicita. L'inferenza su questa porzione fornisce la stessa risposta dell'inferenza sull'intera rete bayesiana implicita.

Un altro metodo per incrementare l'efficienza delle inferenze deriva dalla presenza di sottostrutture ripetute nella rete bayesiana non srotolata. Ciò significa che molti dei fattori costruiti durante l'eliminazione di variabili (e tabelle simili costruite mediante algoritmi di clustering) sono identici; alcuni efficaci schemi di caching hanno consentito velocizzazioni di tre ordini di grandezza per reti di grandi dimensioni.

Terzo, gli algoritmi di inferenza MCMC hanno alcune proprietà interessanti, quando applicati a RPM con incertezza relazionale. MCMC campiona mondi possibili completi, perciò in ogni stato la struttura relazionale è completamente nota. Nell'esempio esposto in precedenza, ogni stato MCMC specificherebbe il valore di $Autore(L_2)$, quindi gli altri potenziali autori non sarebbero più genitori dei nodi dei suggerimenti per L_2 . Per MCMC, quindi, l'incertezza relazionale non aumenta la complessità della rete; il processo MCMC comprende invece transizioni che cambiano la struttura relazionale della rete non srotolata, e di conseguenza la struttura delle sue dipendenze.

Infine, in alcuni casi è possibile evitare del tutto il grounding del modello. I dimostratori di teoremi basati sulla risoluzione e i sistemi di programmazione logica evitano la proposizionalizzazione creando istanze delle variabili logiche solo quando necessario per condurre l'inferenza; in altre parole, “sollevano” (*lift*) il processo di inferenza al di sopra del livello proposizionale ground e ogni passaggio “sollevato” sostituisce molti passaggi ground.

La stessa idea può essere applicata all'inferenza probabilistica. Per esempio, nell'algoritmo di eliminazione di variabili, un fattore “sollevato” può rappresentare un intero insieme di fattori ground che assegnano probabilità a variabili casuali del modello RPM, dove queste variabili casuali differiscono solamente per i simboli di costante utilizzati per costruirle. Illustrare i dettagli di questo metodo va oltre lo scopo di questo libro, ma al termine del capitolo sono indicati alcuni riferimenti.

15.2 Modelli probabilistici a universo aperto

In precedenza abbiamo affermato che la semantica dei database è adeguata per le situazioni in cui conosciamo esattamente l'insieme degli oggetti rilevanti esistenti e possiamo identificarli senza ambiguità (in particolare, tutte le osservazioni su un oggetto sono correttamente associate al simbolo di costante che lo denomina). In molti scenari reali, tuttavia, queste assunzioni sono semplicemente insostenibili. Per esempio, una libreria potrebbe utilizzare un codice ISBN (*international standard book number*) come simbolo di costante per denominare ciascun libro, anche se un determinato libro “logico” (per esempio “Guerra e pace”)

potrebbe avere diversi ISBN corrispondenti a brossura, copertina rigida, stampa a caratteri ingranditi, riedizioni e così via. Sarebbe sensato aggregare i suggerimenti relativi a tutti gli ISBN, ma il negoziante potrebbe non sapere con certezza quali ISBN sono associati a un dato libro (notate che non si tratta di reificare le *singole copie* del libro, cosa che potrebbe essere necessaria per la vendita di libri usati, auto usate e così via). Peggio ancora, ogni cliente è identificato da un ID di accesso ma un cliente disonesto potrebbe avere migliaia di ID! Nel campo della sicurezza informatica gli ID multipli sono detti sybil e il loro utilizzo per ingannare un sistema di tracciamento della reputazione è detto **attacco sybil**.⁵ Quindi, anche una semplice applicazione in un dominio online relativamente ben definito comporta sia **incertezza dell'esistenza** (quali sono i libri e i clienti reali dietro ai dati osservati) sia **incertezza dell'identità** (quali termini logici si riferiscono in realtà allo stesso oggetto).

I fenomeni dell'incertezza dell'esistenza e dell'identità vanno ben al di là delle librerie online. Al contrario, sono onnipresenti:

- Un sistema visivo non sa cosa ci sia dietro l'angolo, e potrebbe non sapere se l'oggetto che vede ora sia lo stesso che ha visto qualche minuto fa.
- Un sistema di analisi di testi non sa in anticipo quali entità appariranno in un testo e deve determinare se espressioni come “Maria”, “Dottoressa Bianchi”, “lei”, “la cardiologa”, “sua madre” e così via si riferiscano allo stesso oggetto.
- Un analista dei servizi segreti a caccia di spie non sa mai quante siano veramente le spie e può solo chiedersi se vari pseudonimi, numeri di telefono e avvistamenti siano legati allo stesso individuo.

In effetti sembra che una parte importante della cognizione umana richieda l'apprendimento di quali oggetti esistono e la capacità di collegare le osservazioni (che non sono quasi mai corredate di ID univoci) agli oggetti del mondo.

Perciò abbiamo la necessità di definire un **modello probabilistico a universo aperto** (OUPM, *open universe probability model*) basato sulla semantica standard della logica del primo ordine, come illustrato nella parte superiore della Figura 15.1. Un linguaggio per i modelli OUPM è un modo semplice per scrivere tali modelli che garantisce nel contempo una distribuzione di probabilità univoca e coerente sullo spazio infinito dei mondi possibili.

attacco sybil
incertezza
dell'esistenza
incertezza
dell'identità

modello
probabilistico a
universo aperto

15.2.1 Sintassi e semantica

L'idea di base è comprendere il modo in cui le reti bayesiane standard e i modelli RPM riescono a definire un modello probabilistico unico e trasferire tale nozione allo scenario del primo ordine. In essenza, una rete bayesiana *genera* tutti i mondi possibili, evento per evento, nell'ordine topologico definito dalla struttura della rete, dove ogni evento è l'assegnamento di un valore a una variabile. Un modello RPM estende tutto ciò a interi insiemi di eventi, definiti dalle possibili istanze delle variabili logiche in un dato predicato o funzione. I modelli OUPM si spingono oltre consentendo passi generativi che *aggiungono oggetti* al mondo possibile che si sta costruendo, dove il numero e il tipo di oggetti potrebbe dipendere dagli oggetti che già si trovano in quel mondo e dalle loro proprietà e relazioni. Ovvero, l'evento che viene generato non è l'assegnamento di un valore a una variabile bensì la stessa *esistenza* di determinati oggetti.

Un modo per fare questo nei modelli OUPM consiste nel fornire **dichiarazioni numeriche** che specificano distribuzioni condizionate sul numero di vari tipi di oggetti. Per esempio, nell'ambito dei suggerimenti sui libri, potremmo voler distinguere tra *clienti* (persone vere)

dichiarazione
numerica

⁵ Il nome “Sybil” deriva da un famoso caso di disturbo di personalità multipla.

e i loro *ID di accesso* (sono in realtà gli ID che danno suggerimenti, non i clienti!). Supponiamo (per semplicità) che il numero di clienti sia uniformemente distribuito tra 1 e 3 e che il numero di libri sia uniformemente distribuito tra 2 e 4:

$$\begin{aligned} \# \text{Cliente} &\sim \text{UniformInt}(1, 3) \\ \# \text{Libro} &\sim \text{UniformInt}(2, 4). \end{aligned} \quad (15.2)$$

Ci aspettiamo che i clienti onesti abbiano solamente un ID, mentre quelli disonesti potrebbero averne un numero qualsiasi tra 2 e 5:

$$\begin{aligned} \# \text{IDAccesso}(\text{Proprietario} = c) &\sim \text{if } \text{Onestà}(c) \text{ then } \text{Esattamente}(1) \\ &\quad \text{else } \text{UniformInt}(2, 5). \end{aligned} \quad (15.3)$$

funzione origine

Questa dichiarazione numerica specifica la distribuzione sul numero degli ID di accesso per i quali il cliente c è il *Proprietario*. La funzione *Proprietario* è detta **funzione origine** perché indica da dove proviene ogni oggetto generato da questa dichiarazione numerica.

distribuzione di Poisson

Il precedente esempio utilizza una distribuzione uniforme sugli interi tra 2 e 5 per specificare il numero degli ID di un cliente disonesto. Questa particolare distribuzione è limitata ma in generale il numero di oggetti potrebbe non essere limitato a priori. La distribuzione utilizzata più frequentemente sugli interi non negativi è la **distribuzione di Poisson**; tale distribuzione ha un parametro λ , che rappresenta il numero atteso di oggetti, e una variabile X campionata da $\text{Poisson}(\lambda)$ ha la distribuzione seguente:

$$P(X = k) = \lambda^k e^{-\lambda} / k!.$$

distribuzione log-normale discreta distribuzione a ordini di grandezza

La varianza della distribuzione di Poisson è a sua volta pari a λ , quindi la deviazione standard è $\sqrt{\lambda}$. Questo significa che per grandi valori di λ la distribuzione è concentrata attorno alla media. Per esempio, se il numero di formiche in un formicaio è modellato da una distribuzione di Poisson con una media di un milione, la deviazione standard è solo mille, ovvero lo 0,1%. Per numeri grandi, spesso ha più senso utilizzare la **distribuzione log-normale discreta**, appropriata quando il logaritmo del numero di oggetti è distribuito normalmente. Una forma particolarmente intuitiva, che chiamiamo **distribuzione a ordini di grandezza**, utilizza i logaritmi in base 10: la distribuzione OdG(3, 1) ha una media di 10^3 e una deviazione standard di un ordine di grandezza; ovvero, gran parte della massa di probabilità ricade tra 10^2 e 10^4 .

variabile numero

La semantica formale dei modelli OUPM inizia con una definizione degli oggetti che popolano i mondi possibili. Nella semantica standard della logica del primo ordine con tipi, gli oggetti sono semplici elementi numerati associati a dei tipi. Nei modelli OUPM, ogni oggetto è una cronologia di generazione; per esempio, un oggetto potrebbe essere “il quarto ID di accesso del settimo cliente” (la ragione di questa costruzione leggermente barocca sarà chiara tra poco). Per i tipi privi di funzione origine, per esempio i tipi *Cliente* e *Libro* dell’Equazione (15.2), gli oggetti hanno origine vuota; per esempio, $\langle \text{Cliente}, , 2 \rangle$ si riferisce al secondo cliente generato dalla dichiarazione numerica. Per le dichiarazioni numeriche con funzione origine, per esempio l’Equazione (15.3), ogni oggetto memorizza la propria origine; per esempio, l’oggetto $\langle \text{IDAccesso}, \langle \text{Proprietario}, \langle \text{Cliente}, , 2 \rangle \rangle, 3 \rangle$ è il terzo ID appartenente al secondo cliente.

Le **variabili numero** di un modello OUPM specificano quanti siano gli oggetti di ciascun tipo per ogni possibile origine e in ogni mondo possibile; quindi $\# \text{IDAccesso}_{\langle \text{Proprietario}, \langle \text{Cliente}, , 2 \rangle \rangle}(\omega) = 4$ significa che, nel mondo ω , il cliente 2 è proprietario di 4 ID di accesso. Come nei modelli relazionali di probabilità, le **variabili casuali di base** determinano i valori dei predicati e delle funzioni per tutte le tuple di oggetti; quindi, $\text{Onestà}_{\langle \text{Cliente}, , 2 \rangle}(\omega) = \text{true}$ significa che nel mondo ω il cliente 2 è onesto. Un mondo possibile è definito dai valori di tutte le variabili numero e variabili casuali base. Un mondo potrebbe essere generato dal modello mediante un campionamento in ordine topologico; la Figura 15.4 ne mostra un esempio. La probabilità di un mondo così costruito è il prodotto delle probabilità di tutti i valori cam-

Variabile	Valore	Probabilità
#Cliente	2	0,3333
#Libro	3	0,3333
Onestà(Cliente, ,1)	true	0,99
Onestà(Cliente, ,2)	false	0,01
Gentilezza(Cliente, ,1)	4	0,3
Gentilezza(Cliente, ,2)	1	0,1
Qualità(Libro, ,1)	1	0,05
Qualità(Libro, ,2)	3	0,4
Qualità(Libro, ,3)	5	0,15
#IDAccesso(Proprietario,(Cliente, ,1))	1	1,0
#IDAccesso(Proprietario,(Cliente, ,2))	2	0,25
Suggerimento(IDAccesso(Proprietario,(Cliente, ,1)),1),(Libro, ,1)	2	0,5
Suggerimento(IDAccesso(Proprietario,(Cliente, ,1)),1),(Libro, ,2)	4	0,5
Suggerimento(IDAccesso(Proprietario,(Cliente, ,1)),1),(Libro, ,3)	5	0,5
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),1),(Libro, ,1)	5	0,4
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),1),(Libro, ,2)	5	0,4
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),1),(Libro, ,3)	1	0,4
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),2),(Libro, ,1)	5	0,4
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),2),(Libro, ,2)	5	0,4
Suggerimento(IDAccesso(Proprietario,(Cliente, ,2)),2),(Libro, ,3)	1	0,4

Figura 15.4 Un particolare mondo per il modello OUPM dei suggerimenti sui libri. Le variabili numero e le variabili casuali di base sono elencate in ordine topologico, assieme ai valori scelti per ognuna di esse e alle probabilità di tali valori.

pionati; in questo caso, $1,2672 \times 10^{-11}$. Ora diventa chiaro perché ogni oggetto contenga la propria origine: questa proprietà assicura che ogni mondo possa essere costruito per mezzo di una e una sola sequenza di generazione. Se così non fosse, la probabilità di un mondo sarebbe una pesante somma combinatoria su tutte le possibili sequenze di generazione che lo creano.

I modelli a universo aperto possono avere una quantità infinita di variabili casuali, quindi la teoria implica considerazioni non banali di teoria della misura. Per esempio, le dichiarazioni numeriche con distribuzione di Poisson o distribuzione a ordini di grandezza consentono quantità illimitate di oggetti, il che conduce a quantità illimitate di variabili casuali per le proprietà e le relazioni di tali oggetti. Inoltre, i modelli OUPM possono avere relazioni di dipendenza ricorsive e tipi infiniti (interi, stringhe, ecc.). Infine, per avere un modello ben formato, sono vietate le dipendenze cicliche e le catene di ascendenti infinite; in generale queste condizioni non sono decidibili, ma determinate condizioni sintattiche sufficienti possono essere verificate facilmente.

15.2.2 Inferenza nei modelli probabilistici a universo aperto

Date le dimensioni potenzialmente enormi e a volte illimitate della rete bayesiana implicita che corrisponde a un tipico modello OUPM, srotolarlo completamente ed effettuare inferenze esatte è poco pratico. Dobbiamo invece considerare algoritmi di inferenza approssimata come MCMC (Paragrafo 13.4.2).

In parole semplici, un algoritmo MCMC per un modello OUPM esplora lo spazio dei mondi possibili definito da insiemi di oggetti e relazioni tra essi, come illustrato nella Figura 15.1 (in alto). Uno spostamento tra stati adiacenti in questo spazio può non solo alterare relazioni e funzioni ma anche aggiungere o sottrarre oggetti e modificare le interpretazioni di simboli di costante. Sebbene ogni mondo possibile possa essere enorme, i calcoli di probabilità richiesti da ogni passaggio (con il campionamento di Gibbs o di Metropolis-Hastings) sono del tutto locali e nella maggior parte dei casi richiedono tempo costante. Ciò perché il

rappporto delle probabilità tra mondi confinanti dipende da un sottografo di dimensioni costanti attorno alle variabili i cui valori vengono modificati. Inoltre, una query logica può essere calcolata in modo *incrementale* in ogni mondo visitato, solitamente in un tempo costante per ogni mondo, invece di ricominciare a calcolare dall'inizio.

Occorre dedicare alcune considerazioni speciali al fatto che un tipico modello OUPM può avere mondi possibili di dimensioni infinite. Consideriamo come esempio il modello del tracciamento di più oggetti della Figura 15.9: la funzione $X(a, t)$, che denota lo stato dell'aereo a al tempo t , corrisponde a una sequenza infinita di variabili per un numero non limitato di aerei a ogni passo. Per questo motivo, MCMC nel caso dei modelli OUPM non campiona mondi possibili completamente specificati, bensì mondi *parziali*, ognuno corrispondente a un insieme disgiunto di mondi possibili. Un mondo parziale è una *istanza minima autocontenuta*⁶ di un sottoinsieme delle variabili *rilevanti*, ovvero ascendenze delle variabili di evidenza e di query. Per esempio, le variabili $X(a, t)$ per valori di t maggiori dell'ultimo tempo di osservazione (o del tempo della query, se maggiore) sono irrilevanti, quindi l'algoritmo può considerare solamente un prefisso finito della sequenza infinita.

15.2.3 Esempi

Il “caso di studio” standard per i modelli OUPM ha tre elementi: il *modello*, le *evidenze* (i fatti noti in un dato scenario) e la *query*, che può essere qualsiasi espressione, eventualmente con variabili logiche libere. La risposta è una probabilità a posteriori congiunta per ogni possibile insieme di sostituzioni per le variabili libere, date le evidenze, secondo il modello.⁷ Ogni modello comprende dichiarazioni dei tipi per i predicati e le funzioni, una o più dichiarazioni numeriche per ogni tipo e una dichiarazione delle dipendenze per ogni predicato e ogni funzione (negli esempi che seguono, le dichiarazioni dei tipi sono omesse dove il significato è chiaro). Come nei modelli RPM, le dichiarazioni delle dipendenze utilizzano la sintassi if-then-else per le dipendenze specifiche di un determinato contesto.

Confronto di citazioni e note bibliografiche

Su Internet si trovano milioni di articoli scientifici e relazioni tecniche in forma di file pdf. Questi testi contengono solitamente un paragrafo, verso la fine, intitolato “Riferimenti” o “Bibliografia”, in cui vengono raccolte citazioni di altri articoli (costituite da stringhe di caratteri) per informare il lettore sui lavori collegati. Queste stringhe possono essere individuate e raccolte dai file pdf allo scopo di creare una rappresentazione di tipo database che metta in relazione articoli accademici e ricercatori tramite collegamenti bibliografici. Sistemi come CiteSeer e Google Scholar offrono ai loro utenti una tale rappresentazione; dietro le quinte, alcuni algoritmi lavorano per trovare gli articoli, estrarre le stringhe bibliografiche e identificare gli articoli a cui le citazioni si riferiscono. È un compito difficile perché queste stringhe non contengono identificatori degli oggetti e contengono invece errori di sintassi o di ortografia, punteggiatura e contenuto. Per illustrare tutto ciò, ecco due esempi relativamente innocui:

1. [Lashkari et al 94] Collaborative Interface Agents, Yezdi Lashkari, Max Metral, and Pattie Maes, Proceedings of the Twelfth National Conference on Artificial Intelligence, MIT Press, Cambridge, MA, 1994.
2. Metral M. Lashkari, Y. and P. Maes. Collaborative interface agents. In Conference of the American Association for Artificial Intelligence, Seattle, WA, August 1994.

⁶ Un'istanza autocontenuta di un insieme di variabili è un'istanza tale che i genitori di ogni variabile dell'insieme fanno a loro volta parte dell'insieme.

⁷ Come con Prolog, potrebbero esserci infiniti insiemi di sostituzioni di dimensioni non limitate; progettare interfacce di esplorazione per queste risposte è un'interessante sfida di visualizzazione.

La questione fondamentale riguarda l'identità: sono citazioni dello stesso articolo accademico o di articoli differenti? Di fronte a questa domanda, anche gli esperti discordano o sono incapaci di decidere, indicando che il ragionamento in condizioni di incertezza è destinato a essere una parte importante della soluzione di questo problema.⁸ Gli approcci *ad hoc*, come i metodi basati su criteri di somiglianza testuale, spesso falliscono miseramente. Per esempio, nel 2002 CiteSeer segnalava oltre 120 diversi libri scritti da Russell e Norvig.

Per risolvere questo problema utilizzando un approccio probabilistico, ci occorre un modello generativo per il dominio. In altre parole, ci chiediamo come nascano queste stringhe bibliografiche. Il processo inizia con i ricercatori, che hanno dei nomi (non ci occupiamo di come i ricercatori siano venuti al mondo; dobbiamo solo esprimere la nostra incertezza su quanti siano). Questi ricercatori hanno scritto degli articoli scientifici, che hanno dei titoli; le persone citano gli articoli, combinando i nomi degli autori e i titoli degli articoli (con errori) nel testo della citazione bibliografica secondo certe regole. Gli elementi base di questo modello sono mostrati nella Figura 15.5, che tratta il caso in cui ogni articolo ha solo un autore.⁹

Date solamente delle stringhe bibliografiche come evidenze, l'inferenza probabilistica su questo modello per selezionare la spiegazione più probabile dei dati produce un tasso di errore 2 o 3 volte inferiore rispetto a quello di CiteSeer (Pasula *et al.*, 2003). Il processo di inferenza mostra anche una forma di disambiguazione collettiva, fondata sulla conoscenza: più numerose sono le citazioni di un determinato articolo, più accuratamente ognuna di esse è riconosciuta, perché il riconoscimento deve concordare con i fatti riguardanti l'articolo.

```

type Ricercatore, Articolo, Citazione
random String Nome(Ricercatore)
random String Titolo(Articolo)
random Articolo PubCited(Citazione)
random String Testo(Citazione)
random Boolean Professore(Ricercatore)
origin Ricercatore Autore(Articolo)

#Ricercatore ~ OdG(3, 1)
Nome(r) ~ APrioriNome()
Professore(r) ~ Boolean(0,2)
#Articolo(Autore = r) ~ if Professore(r) then OdG(1,5, 0,5) else OdG(1, 0,5)
Titolo(a) ~ APrioriTitoloArticolo()
ArticoloCitato(c) ~ UniformChoice({Articolo a})
Testo(c) ~ GrammaticaHMM(Nome(Autore(ArticoloCitato(c))), Titolo(ArticoloCitato(c)))

```

Figura 15.5 Un modello OUPM per l'estrazione delle informazioni bibliografiche. Per semplicità il modello assume che ogni articolo abbia un solo autore e omette i dettagli dei modelli di grammatica e di errore.

⁸ La risposta è sì, è lo stesso articolo scientifico. La “National Conference on Artificial Intelligence” (notare l’assenza di “fi” in Artificial, per un errore nell’interpretazione della legatura dei caratteri) è un altro nome della conferenza AAAI; la conferenza ha avuto luogo a Seattle mentre gli atti sono stati pubblicati da un editore di Cambridge.

⁹ Il caso multi-autore ha la stessa struttura generale ma è un po’ più complicato. Le parti del modello non mostrate (*APrioriNome*, *APrioriTitoloArticolo* e *GrammaticaHMM*) sono modelli probabilistici tradizionali. Per esempio, *APrioriNome* è un mix di una distribuzione categoriale su nomi veri e di un modello a trigrammi (cfr. Paragrafo 23.1 del Volume 2) per trattare i nomi mai incontrati in precedenza, entrambi addestrati con dati del database anagrafico statunitense U.S. Census.

```

#EventiSismici ~ Poisson( $T * \lambda_e$ )
Tempo(e) ~ UniformReal(0,  $T$ )
Terremoto(e) ~ Boolean(0.999)
Posizione(e) ~ if Terremoto(e) then APrioriSpaziale() else UniformEarth()
Profondità(e) ~ if Terremoto(e) then UniformReal(0, 700) else Esattamente(0)
Magnitudo(e) ~ Exponential(log(10))
Rilevato(e, f, s) ~ Logistic(pesi(s, f), Magnitudo(e), Profondità(e), Dist(e, s))
#Rilevamenti(sito = s) ~ Poisson( $T * \lambda_f(s)$ )
#Rilevamenti(evento=e, fase=f, stazione=s) = if Rilevato(e, f, s) then 1 else 0
TempoInizio(a, s) if (evento(a) = null) then ~ UniformReal(0,  $T$ )
else = Tempo(evento(a)) + GeoTT(Dist(evento(a), s), Profondità(evento(a)), fase(a))
      + Laplace( $\mu_t(s)$ ,  $\sigma_t(s)$ )
Ampiezza(a, s) if (evento(a) = null) then ~ ModelloAmpRumoroso(s)
else = ModelloAmp(Magnitudo(evento(a)), Dist(evento(a), s), Profondità(evento(a)), fase(a))
Azimut(a, s) if (evento(a) = null) then ~ UniformReal(0, 360)
else = GeoAzimut(Posizione(evento(a)), Profondità(evento(a)), fase(a), Sito(s))
      + Laplace(0,  $\sigma_a(s)$ )
Lentezza(a, s) if (evento(a) = null) then ~ UniformReal(0, 20)
else = LentezzaGeo(Posizione(evento(a)), Profondità(evento(a)), fase(a), Sito(s))
      + Laplace(0,  $\sigma_s(s)$ )
FaseOsservata(a, s) ~ ModelloFaseCategorico(fase(a))

```

Figura 15.6 Una versione semplificata del modello NET-VISA (vedere il testo).

Monitoraggio del trattato nucleare

La verifica del rispetto del trattato sul divieto di test nucleari (CTBTO, *comprehensive nuclear-test-ban treaty*) richiede che vengano rilevati tutti gli eventi sismici sulla Terra al di sopra di una determinata magnitudo. L’organizzazione internazionale delle Nazioni Unite (ONU) che si occupa del controllo (UNCTBTO) utilizza una rete di sensori chiamata IMS (*international monitoring system*); il suo software di elaborazione automatica, basato su 100 anni di ricerca sismologica, ha un tasso di mancati rilevamenti di circa il 30%. Il sistema NET-VISA (Arora *et al.*, 2013), basato su un modello OUPM, riduce significativamente questi errori.

Il modello NET-VISA (Figura 15.6) esprime in modo diretto i dati geofisici rilevanti. Descrive distribuzioni sul numero di eventi in un dato intervallo di tempo (la maggior parte dei quali è di origine naturale) nonché sul momento in cui si verificano e sulla loro magnitudo, profondità e posizione. Le posizioni degli eventi naturali hanno una distribuzione spaziale a priori ricavata (come altre parti del modello) dai dati storici; secondo le regole del trattato, si assume che gli eventi di origine umana si verifichino uniformemente sulla superficie terrestre. A ogni stazione s , ogni fase (tipo di onda sismica) f derivante da un evento e produce 0 oppure 1 rilevamenti (segnali sopra la soglia); la probabilità del rilevamento dipende dalla magnitudo e dalla profondità dell’evento e dalla sua distanza dalla stazione. Anche i “falsi allarmi” si verificano secondo un parametro legato alla stazione. L’istante di arrivo, l’ampiezza e altre proprietà di un rilevamento r derivante da un evento reale dipendono dalle proprietà dell’evento originante e dalla sua distanza dalla stazione.

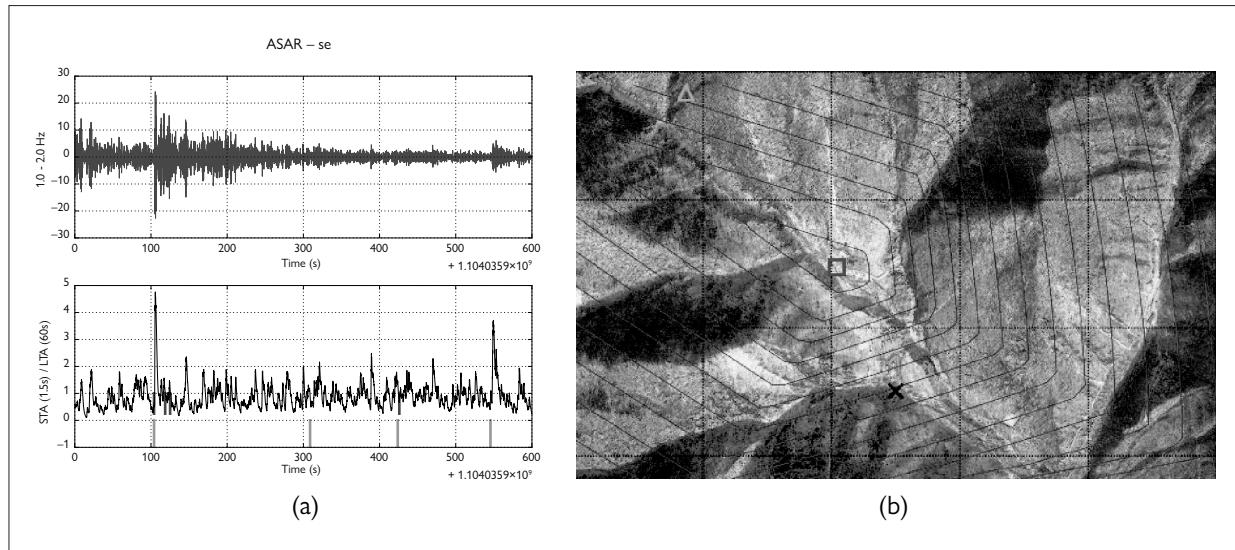


Figura 15.7 (a) In alto: esempio di forma d'onda registrata ad Alice Springs, in Australia. In basso: la forma d'onda dopo l'elaborazione per rilevare i tempi di arrivo delle onde sismiche. Le linee scure sono gli arrivi rilevati automaticamente; le linee chiare sono gli arrivi effettivi. (b) Stime della posizione del test nucleare nordcoreano del 12 febbraio 2013: UNCTBTO Late Event Bulletin (triangolo chiaro in alto a sinistra); NET-VISA (quadrato al centro). L'ingresso del luogo di test sotterraneo (la piccola "x") si trova a 0,75 km dalla stima di NET-VISA. I contorni mostrano la distribuzione di probabilità a posteriori di NET-VISA. Cortesia di CTBTO Preparatory Commission.

Una volta addestrato, il modello funziona in modo continuativo. Le evidenze sono costituite dai rilevamenti (90% dei quali sono falsi allarmi) estratti dai dati grezzi dell'IMS sulle forme d'onda, e la query chiede tipicamente la cronologia degli eventi, o *bollettino*, più probabile considerati i dati. I risultati finora ottenuti sono incoraggianti; per esempio, nel 2009 il bollettino automatizzato delle Nazioni Unite SEL3 mancò di rilevare il 27,4% dei 27.294 eventi nell'intervallo di magnitudo 3–4 mentre NET-VISA ne perse l'11,1%. Inoltre, i confronti con reti regionali dense mostrano che NET-VISA trova fino al 50% in più di eventi reali rispetto ai bollettini prodotti dagli analisti sismici esperti dell'ONU. NET-VISA tende anche ad associare più rilevamenti a un dato evento, portando a stime più precise della posizione (vedere Figura 15.7). L'1 gennaio 2018 il modello NET-VISA è stato attivato nell'ambito del processo di monitoraggio della CTBTO.

Malgrado le superficiali differenze, i due esempi sono strutturalmente simili: ci sono oggetti sconosciuti (pubblicazioni, terremoti) che generano percezioni secondo un determinato processo fisico (citazione, propagazione sismica). Le percezioni sono ambigue, rispetto alla loro origine, ma quando si ipotizza che più percezioni siano originate dallo stesso oggetto sconosciuto, le proprietà di quell'oggetto possono essere inferite con maggiore precisione.

La stessa struttura e gli stessi schemi di ragionamento valgono per ambiti come l'eliminazione di duplicati nei database e la comprensione del linguaggio naturale. In alcuni casi, inferire l'esistenza di un oggetto implica il raggruppamento delle percezioni, un processo che assomiglia al clustering dell'apprendimento automatico. In altri casi, un oggetto può non generare alcuna percezione e tuttavia la sua esistenza può essere inferita, come accadde per esempio quando le osservazioni di Urano portarono alla scoperta di Nettuno. L'esistenza dell'oggetto inosservato si deduce dai suoi effetti sul comportamento e sulle proprietà di oggetti osservati.

15.3 Tenere traccia di un mondo complesso

Nel Capitolo 14 ci siamo occupati del problema di come tenere traccia dello stato del mondo, considerando però solo i casi delle rappresentazioni atomiche (HMM) e delle rappresentazioni fattorizzate (DBN e filtri di Kalman). Ciò ha senso per i mondi con un singolo oggetto, per esempio un singolo paziente del reparto di terapia intensiva o un singolo uccello che vola attraverso la foresta. In questo paragrafo vediamo cosa accade quando le osservazioni sono generate da due o più oggetti. Ciò che distingue questo caso dalla vecchia semplice stima dello stato è che ora esiste la possibilità dell'*incertezza* su quale oggetto abbia creato una determinata osservazione. È il problema dell'**incertezza dell'identità** del Paragrafo 15.2, visto ora in un contesto temporale. Nella letteratura sulla teoria del controllo è il problema dell'**associazione dei dati**, ovvero il problema di associare i dati osservati agli oggetti che li hanno generati. Lo si potrebbe considerare un ulteriore esempio di modellazione probabilistica a universo aperto, ma ha un'importanza pratica tale da meritare un proprio paragrafo.

15.3.1 Esempio: tracciamento multitarget

Il problema dell'associazione dei dati è stato studiato inizialmente nel contesto del rilevamento radar di più oggetti, nel quale gli impulsi riflessi vengono rilevati a intervalli temporali fissi da un'antenna radar rotante. A ogni passo temporale, sullo schermo potrebbero apparire più segnali, ma non esiste alcuna osservazione diretta che indichi quale segnale al tempo t corrisponda a un determinato segnale al tempo $t - 1$. La Figura 15.8(a) mostra un semplice esempio con due segnali a ogni passo temporale, per cinque passi. Ogni segnale è associato al tempo corrispondente ma è privo di informazioni di identificazione.

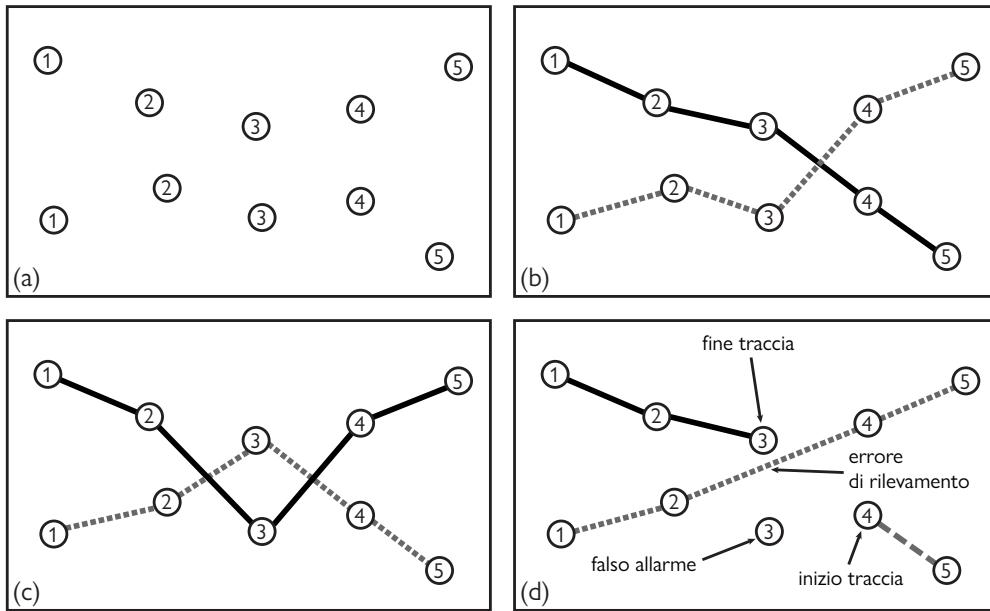


Figura 15.8 (a) Osservazioni delle posizioni di alcuni oggetti in uno spazio 2D, in cinque momenti. Ogni osservazione è contrassegnata con il riferimento temporale ma non identifica l'oggetto che l'ha prodotta. (b–c) Ipotesi sui percorsi degli oggetti. (d) Un'ipotesi per il caso in cui sono possibili falsi allarmi, mancati rilevamenti e inizio/fine delle tracce.

Ipotizziamo, per il momento, di sapere che i segnali siano generati da esattamente due aeromobili, A_1 e A_2 . Nella terminologia degli OUPM, A_1 e A_2 sono **oggetti garantiti**, nel senso che è garantito che esistano e che siano distinti; inoltre, in questo caso, non ci sono altri oggetti (in altre parole, per quanto riguarda i velivoli, questo scenario corrisponde alla semantica dei database presupposta dai RPM). Indichiamo le loro vere posizioni con $X(A_1, t)$ e $X(A_2, t)$, dove t è un intero non negativo che indica i tempi di aggiornamento del sensore. Assumiamo che la prima osservazione arrivi a $t = 1$ e che al tempo 0 la distribuzione a priori per la posizione di ogni aeromobile sia $\text{Init}X()$. Per semplicità, ipotizziamo inoltre che ogni aereo si muova indipendentemente secondo un modello di transizione noto, per esempio un modello gaussiano lineare come quello utilizzato nel filtro di Kalman (Paragrafo 14.4).

L'elemento finale è il modello sensoriale: assumiamo nuovamente un modello gaussiano lineare in cui un aeromobile nella posizione \mathbf{x} produce un segnale s la cui posizione osservata $Z(s)$ è una funzione lineare di x con l'aggiunta di rumore gaussiano. Ogni aeromobile genera uno e un solo segnale a ogni passo temporale, quindi il segnale ha come origine un aereo e un tempo. Quindi, tralasciando per ora la probabilità a priori, il modello appare così:

guaranteed Aeromobile A_1, A_2
 $X(a, t) \sim \text{if } t = 0 \text{ then } \text{Init}X() \text{ else } \mathcal{N}(\mathbf{F} X(a, t - 1), \Sigma_x)$
 $\#\text{Segnale}(\text{Origine}=a, \text{Tempo}=t) = 1$
 $Z(s) \sim \mathcal{N}(\mathbf{H} X(\text{Origine}(s), \text{Tempo}(s)), \Sigma_z)$

dove \mathbf{F} e Σ_x sono matrici che descrivono il modello di transizione lineare e la covarianza del rumore delle transizioni, mentre \mathbf{H} e Σ_z sono le corrispondenti matrici per il modello sensoriale (cfr. Paragrafo 14.4.3).

La differenza fondamentale tra questo modello e un filtro di Kalman standard è che le letture del sensore (segnali) sono prodotte da *due* oggetti. Ciò significa che a ogni passo temporale c'è *incertezza* riguardo a quale oggetto abbia generato ciascuna lettura. Ogni mondo possibile di questo modello comprende un'associazione tra aeromobile e segnali, definita dai valori di tutte le variabili $\text{Origine}(s)$ per tutti i passi temporali. La Figura 15.8(b–c) mostra due ipotesi di associazione. In generale, per n oggetti e T passi temporali, esistono $(n!)^T$ modi per assegnare i segnali ai velivoli: un numero tremendamente grande.

Lo scenario descritto fin qui coinvolge n oggetti noti che generano n osservazioni a ogni passo temporale. Le applicazioni concrete dell'associazione di dati sono tipicamente molto più complicate. Spesso le osservazioni riportate comprendono **falsi allarmi** (detti anche **clutter**), che non sono causati da oggetti reali. Possono esserci **mancati rilevamenti**, nel senso che per un oggetto reale non vengono riportate osservazioni. Infine, appaiono nuovi oggetti e ne scompaiono altri. Questi fenomeni, che creano ancor più mondi possibili da prendere in considerazione, sono illustrati nella Figura 15.8(d). Il corrispondente modello OUPM è mostrato nella Figura 15.9.

Data la sua importanza pratica in applicazioni sia civili sia militari, il problema del tracciamento di più oggetti e dell'associazione dei dati è stato affrontato in decine di migliaia di articoli accademici. Molti di essi tentano semplicemente di trattare i complessi dettagli matematici dei calcoli di probabilità per il modello della Figura 15.9, o per sue versioni più semplici. In un certo senso ciò non è necessario una volta che il modello è espresso in un linguaggio di programmazione probabilistica, perché il motore di inferenza generico gestisce correttamente gli aspetti matematici per qualsiasi modello, compreso questo. Inoltre, altre varianti dello scenario (volo in formazione, oggetti diretti verso destinazioni sconosciute, oggetti che decollano o atterrano e così via) possono essere gestite con piccole modifiche del modello senza ricorrere a nuove derivazioni matematiche né a programmazione complessa.

Dal punto di vista pratico, la sfida posta da questo tipo di modello è la complessità dell'inferenza. Come in ogni altro modello di probabilità, inferenza significa escludere le varia-

oggetto garantito

falso allarme
clutter
mancato
rilevamento

```

#Aeromobile(TempoEntrata =t) ~ Poisson(λa)
Esce(a, t) ~ if InVolo(a, t) then Boolean(αe)
InVolo(a, t) = (t=TempoEntrata(a)) ∨ (InVolo(a, t - 1) ∧ ¬ Esce(a, t - 1))
X(a, t) ~ if t = TempoEntrata(a) then InitX()
else if InVolo(a, t) then N(FX(a, t - 1), Σx)
#Segnale(Origine=a, Time=t) ~ if InVolo(a, t) then Bernoulli(ProbRilevamento(X(a, t)))
#Segnale(Tempo=t) ~ Poisson(λf)
Z(s) ~ if Origine(s)=null then UniformZ(R) else N(H X(Origine(s), Tempo(s)), Σz)

```

Figura 15.9 Un modello OUPM per il tracciamento radar di più oggetti, con falsi allarmi, mancati rilevamenti ed entrata e uscita degli aeromobili. Il tasso con cui appaiono nuovi aeromobili è λ_a , mentre la probabilità per passo temporale che un aereo esca di scena è α_e . I falsi allarmi (ovvero segnali non prodotti da un aereo) appaiono uniformemente nello spazio al ritmo di λ_f per passo temporale. La probabilità che un aeromobile venga rilevato (cioè che produca un segnale) dipende dalla sua posizione corrente.

bili che non siano la query e le evidenze. Per il filtraggio in HMM e DBN abbiamo potuto escludere le variabili di stato da 1 a $t - 1$ con un semplice trucco di programmazione dinamica; per i filtri di Kalman abbiamo sfruttato speciali proprietà delle gaussiane. Per l'associazione dei dati siamo meno fortunati. Non è noto alcun algoritmo esatto ed efficiente, per lo stesso motivo per cui non ne esistono per il filtro di Kalman a commutazione (Paragrafo 14.4.4): la distribuzione del filtro, che descrive la distribuzione congiunta su numero e posizioni degli aeromobili a ogni passo temporale, diventa un mix di un numero esponenziale di distribuzioni, una per ogni modo di selezionare una sequenza di osservazioni da assegnare a ciascun aereo.

Per affrontare la complessità dell'inferenza esatta sono stati utilizzati diversi metodi approssimati. L'approccio più semplice consiste nella scelta di un unico assegnamento “migliore” a ogni passo temporale, date le posizioni predette degli oggetti al tempo attuale. Questo metodo associa osservazioni e oggetti e permette di tracciare il percorso di ogni oggetto e di effettuare una predizione per il passo temporale successivo. Per scegliere l'assegnamento “migliore” è comune utilizzare il cosiddetto **filtro del vicino più prossimo**, che sceglie ripetutamente la posizione predetta e l'osservazione che risultano più vicine tra loro, e aggiunge tale coppia all'assegnamento. Il metodo del vicino più prossimo funziona bene quando gli oggetti sono ben separati nello spazio e l'incertezza della predizione e l'errore di osservazione sono modesti; in altre parole, quando non c'è possibilità di confusione.

Quando c'è maggiore incertezza riguardo al corretto assegnamento, un approccio migliore consiste nello scegliere l'assegnamento che massimizza la probabilità congiunta delle osservazioni correnti date le posizioni predette. Lo si può fare in modo efficiente ricorrendo all'**algoritmo ungherese** (Kuhn, 1955), anche quando a ogni passo temporale esistono $n!$ assegnamenti tra cui scegliere.

I metodi che si affidano al migliore assegnamento per ogni passo temporale falliscono miseramente quando le condizioni sono più difficili. In particolare, se l'algoritmo si affida a un assegnamento non corretto, la predizione per il passo temporale successivo può essere significativamente sbagliata, il che conduce a successivi assegnamenti ancora meno corretti, e così via. Gli approcci con campionamento possono essere molto più efficaci. L'algoritmo di **particle filtering** (letteralmente “filtraggio con particelle”, Paragrafo 14.5.3) per l'associazione dei dati mantiene una vasta raccolta di possibili assegnamenti. L'algoritmo **MCMC** esplora lo spazio delle cronologie di assegnamento (per esempio, la Figura 15.8(b–c)) potrebbe riferirsi a stati nello spazio degli stati MCMC) e può cambiare idea sulle decisioni di assegnamento precedenti.

filtro del vicino più prossimo

algoritmo ungherese



Figura 15.10 Immagini da due videocamere di sorveglianza, (a) a monte e (b) a valle, distanti l'una dall'altra circa tre chilometri lungo la Highway 99 a Sacramento, in California. Il veicolo nel riquadro è stato identificato con entrambe le videocamere.

Un modo ovvio per velocizzare l'inferenza basata sul campionamento per il tracciamento di più oggetti consiste nell'utilizzare la tecnica di **Rao-Blackwell** del Capitolo 14 (Paragrafo 14.5.3): data una specifica ipotesi di associazione di tutti gli oggetti, è generalmente possibile effettuare il calcolo del filtraggio per ciascun oggetto in modo esatto ed efficiente, invece di campionare molte possibili sequenze di stati degli oggetti. Per esempio, con il modello della Figura 15.9, il calcolo del filtraggio comporta solo l'utilizzo di un filtro di Kalman per la sequenza di osservazioni assegnata a un dato oggetto. Inoltre, quando si passa da un'ipotesi di associazione a un'altra, è necessario ripetere i calcoli solo per gli oggetti per i quali l'insieme delle osservazioni associate è stato modificato. Gli attuali metodi MCMC per l'associazione dei dati possono gestire molte centinaia di oggetti in tempo reale e fornire una buona approssimazione delle vere distribuzioni a posteriori.

15.3.2 Esempio: monitoraggio del traffico

La Figura 15.10 mostra due immagini provenienti da due videocamere distanti tra loro, lungo un'autostrada californiana. In questa applicazione siamo interessati a due obiettivi: stimare il tempo necessario, nelle attuali condizioni di traffico, per spostarsi da un luogo all'altro sulla rete autostradale; e misurare la *domanda*, ovvero il numero dei veicoli che viaggiano tra due punti qualsiasi della rete in determinati momenti della giornata e in particolari giorni della settimana. Entrambi gli obiettivi richiedono la soluzione del problema dell'associazione dei dati su un'area estesa con molte videocamere e decine di migliaia di veicoli l'ora.

Nella sorveglianza visiva, i falsi allarmi sono provocati da ombre in movimento, veicoli articolati, riflessi nelle pozanghere e così via; i mancati rilevamenti sono provocati da occlusioni, nebbia, oscurità e mancanza di contrasto visivo; inoltre i veicoli entrano ed escono costantemente dalla rete autostradale in punti che potrebbero non essere monitorati. Per di più, l'aspetto di un dato veicolo può cambiare notevolmente tra due videocamere a seconda delle condizioni di illuminazione e della posizione del veicolo nell'immagine, e il modello di transizione cambia con il formarsi e il risolversi degli intasamenti. Infine, con un traffico denso e con videocamere molto distanti, l'errore di predizione del modello di transizione per un'auto che si muove tra una videocamera e l'altra è molto maggiore della tipica distanza tra i veicoli. Malgrado questi problemi, alcuni moderni algoritmi di associazione dei dati sono stati in grado di stimare correttamente i parametri del traffico in scenari reali.

L’associazione dei dati è fondamentale per il monitoraggio di un mondo complesso, perché senza di essa non c’è modo di combinare tra loro più osservazioni di un dato oggetto. Quando gli oggetti di un mondo interagiscono fra loro in attività complesse, per comprendere quel mondo occorre combinare l’associazione dei dati con i modelli relazionali di probabilità e a universo aperto del Paragrafo 15.2. Questa è un’area di ricerca attualmente attiva.

15.4 Programmi come modelli probabilistici

Molti linguaggi di programmazione probabilistica sono stati costruiti basandosi sull’idea che i modelli probabilistici possano essere definiti utilizzando codice scritto in qualsiasi linguaggio di programmazione che comprenda una fonte di casualità. Per tali modelli, i mondi possibili sono tracciati di esecuzione e la probabilità di un tracciato è quella delle scelte casuali necessarie affinché esso si verifichi. I PPL creati in questo modo ereditano tutta la capacità espressiva del linguaggio sottostante, comprese le strutture di dati complesse, la ricorsività e, in alcuni casi, le funzioni di ordine superiore. Molti PPL sono in effetti computazionalmente universali: possono rappresentare ogni distribuzione di probabilità che possa essere campionata da una macchina di Turing che termina.

15.4.1 Esempio: lettura di testo

Illustriamo questo approccio alla modellazione e inferenza probabilistica mediante il problema di scrivere un programma che legga un testo alterato. Modelli di questo genere possono essere destinati a leggere testi sbavati o indistinti perché danneggiati dall’acqua, oppure chiazzati per l’invecchiamento della carta su cui sono stampati. Possono anche essere costruiti allo scopo di aggirare alcuni tipi di verifica CAPTCHA.

La Figura 15.11 mostra una programma generativo contenente due componenti: (i) un sistema per generare sequenze di lettere e (ii) un sistema per generare un’immagine disturbata e sfocata di queste lettere utilizzando una libreria grafica. La Figura 15.12 (in alto) mostra degli esempi di immagini generate richiamando nove volte la funzione GENERA-IMMAGINE.

15.4.2 Sintassi e semantica

programma generativo

tracciato di esecuzione

Un **programma generativo** è un programma eseguibile in cui ogni scelta casuale definisce una variabile casuale in un modello probabilistico. Immaginiamo di srotolare l’esecuzione di un programma che compie scelte casuali, passo per passo. Sia X_i la variabile casuale corrispondente alla i -esima scelta casuale compiuta dal programma; come di consueto, x_i denota un possibile valore di X_i . Chiamiamo $\omega = \{x_i\}$ un **tracciato di esecuzione** del programma generativo, ovvero una sequenza di possibili valori per le scelte casuali. Ogni esecuzione del programma genera un tale tracciato; da ciò il termine “programma generativo”.

Lo spazio di tutti i possibili tracciati di esecuzione Ω può essere visto come lo spazio di campionamento di un modello probabilistico definito dal programma generativo. La distribuzione di probabilità sui tracciati può essere definita come il prodotto delle probabilità delle singole scelte casuali: $P(\omega) = \prod_i P(x_i | x_1, \dots, x_{i-1})$. Questo è analogo alla distribuzione sui mondi di un modello OUPM.

È concettualmente immediato convertire qualsiasi modello OUPM in un corrispondente programma generativo. Questo programma generativo effettua scelte casuali per ogni dichiarazione numerica e per il valore di ogni variabile casuale di base la cui esistenza converge dalle dichiarazioni numeriche. Il principale lavoro aggiuntivo che il programma generativo deve fare è creare strutture di dati che rappresentino gli oggetti, le funzioni e le relazioni dei possibili mondi del modello OUPM. Queste strutture di dati vengono create automaticamente dal motore di inferenza dell’OUPM perché tale modello assume che ogni

```

function GENERA-IMMAGINE() returns un'immagine con delle lettere
    lettere  $\leftarrow$  GENERA-LETTERE(10)
    return RENDER-IMMAGINE-ALTERATA(lettere, 32, 128)
function GENERA-LETTERE( $\lambda$ ) returns un vettore di lettere
    n  $\sim$  Poisson( $\lambda$ )
    letters $\leftarrow$  []
    for i = 1 to n do
        lettere[i]  $\sim$  UniformChoice({a,b,c,· · ·})
    return lettere
function RENDER-IMMAGINE-ALTERATA(lettere, larghezza, altezza) returns un'immagine alterata delle lettere
    immagine_pulita  $\leftarrow$  RENDER(lettere, larghezza, altezza, testo_su = 10, testo_sin = 10)
    immagine_alterata  $\leftarrow$  []
    varianza_alterazione  $\sim$  UniformReal(0,1, 1)
    for riga = 1 to larghezza do
        for col = 1 to altezza do
            immagine_alterata[riga,col]  $\sim$   $\mathcal{N}$ (immagine_pulita[riga,col], varianza_alterazione)
    return immagine_alterata

```

Figura 15.11 Programma generativo per un modello probabilistico a universo aperto per il riconoscimento ottico dei caratteri. Il programma produce immagini alterate di sequenze di lettere: genera una sequenza, la converte in un'immagine 2D e poi inserisce del rumore aggiuntivo per ciascun pixel.

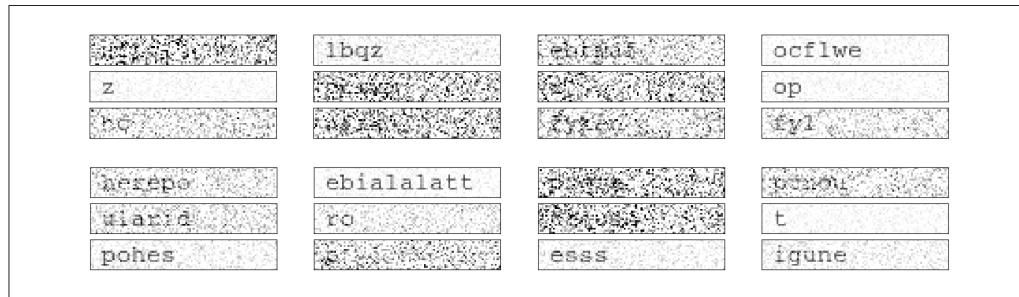


Figura 15.12 Le dodici immagini in alto sono state prodotte dal programma generativo della Figura 15.11. Il numero di lettere, le lettere stesse, la quantità di rumore aggiunto e lo specifico rumore a livello di pixel fanno tutti parte del dominio del modello probabilistico. Le dodici immagini in basso sono state prodotte dal programma generativo della Figura 15.15. Il modello di Markov genera tipicamente sequenze di lettere più semplici da pronunciare.

mondo possibile sia una struttura di modello del primo ordine, mentre tipicamente questa ipotesi manca nei PPL.

Le immagini della Figura 15.12 possono favorire una comprensione intuitiva della distribuzione di probabilità $P(\Omega)$: vediamo vari livelli di rumore e, nelle immagini meno disturbate, vediamo anche sequenze di lettere di varia lunghezza. Sia ω_1 il tracciato corrispondente all'immagine nell'angolo superiore destro di questa figura, contenente le lettere *ocflwe*. Se srotolassimo questo tracciato ω_1 per ottenere una rete bayesiana, avrebbe 4104 nodi: 1 per la variabile *n*; 6 per le variabili *lettere*[*i*]; 1 per la *varianza del rumore*; e 4096 nodi per i pixel dell'*immagine_alterata*. Vediamo quindi che questo programma generativo definisce

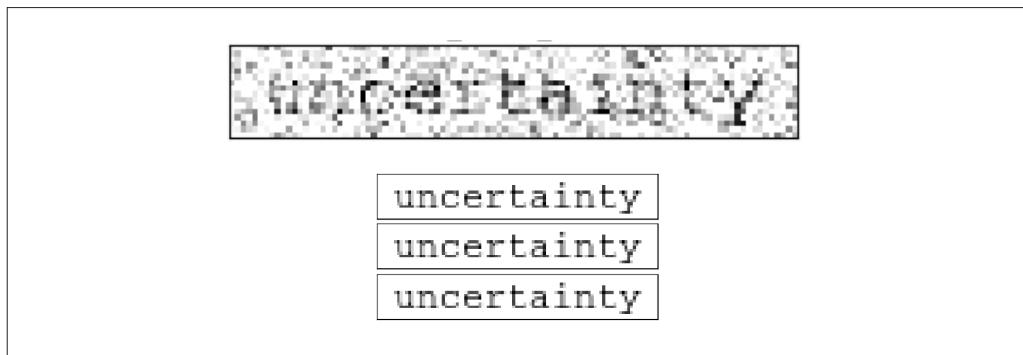


Figura 15.13 Immagine di input alterata (in alto) e risultati dell'inferenza (in basso) prodotti da tre esecuzioni con 25 iterazioni MCMC ciascuna, con il modello della Figura 15.11. Notate che il processo di inferenza identifica correttamente la sequenza di lettere.

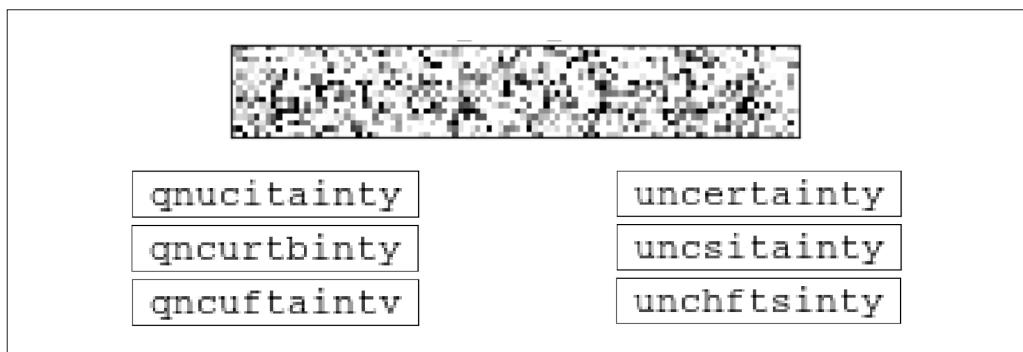


Figura 15.14 In alto: immagine di input estremamente alterata. In basso a sinistra: tre risultati di inferenze con 25 iterazioni MCMC con il modello della Figura 15.11, in cui le lettere sono indipendenti. In basso a destra: tre risultati di inferenze con il modello a digrammi della Figura 15.15. Con entrambi i modelli i risultati contengono ambiguità ma quelli del secondo modello riflettono la conoscenza a priori di sequenze di lettere plausibili.

un modello probabilistico a universo aperto: il numero di scelte casuali che compie non è limitato a priori, bensì dipende dal valore della variabile casuale n .

15.4.3 Risultati delle inferenze

Utilizziamo questo modello per interpretare immagini di lettere alterate con rumore di tipo additivo. La Figura 15.13 mostra un'immagine disturbata e i risultati di tre esecuzioni indipendenti di MCMC. Per ogni esecuzione, mostriamo un'immagine delle lettere contenute nel tracciato dopo avere interrotto la catena di Markov. In tutti e tre i casi il risultato è la sequenza di lettere *uncertainty*, che suggerisce che la distribuzione a posteriori sia fortemente concentrata sull'interpretazione corretta.

Alteriamo ulteriormente il testo, sfocandolo quanto basta per renderne difficile la lettura da parte di una persona. La Figura 15.14 mostra i risultati delle inferenze su questo input più ostico. Questa volta, sebbene l'inferenza MCMC sembri convergere sul numero di lettere corretto (che noi conosciamo), la prima lettera viene identificata erroneamente come una *q* e c'è incertezza su cinque delle dieci lettere seguenti.

A questo punto, esistono molti modi possibili per interpretare i risultati. Può darsi che l'inferenza MCMC abbia funzionato bene e che i risultati riflettano adeguatamente la vera

probabilità a posteriori, dati il modello e l'immagine; in questo caso, l'incertezza su alcune lettere e l'errore sulla prima sono inevitabili. Per ottenere risultati migliori, potrebbe essere necessario migliorare il modello di testo o ridurre il livello di rumore. È anche possibile che l'inferenza MCMC non abbia funzionato adeguatamente: se eseguissimo 300 catene per 25.000 o 25 milioni di iterazioni, potremmo trovare una distribuzione dei risultati piuttosto differente, che indicherebbe forse che la prima lettera è probabilmente una u e non una q.

Eseguire più processi di inferenza potrebbe essere costoso in termini economici e di tempo. Inoltre, non esistono test infallibili per la convergenza dei metodi di inferenza Monte Carlo. Potremmo tentare di migliorare l'algoritmo di inferenza, per esempio progettando una migliore distribuzione delle proposte per MCMC o utilizzando indizi bottom-up (dal basso verso l'alto) dall'immagine per suggerire ipotesi iniziali migliori. Queste migliorie richiedono ulteriori ragionamenti, implementazione e correzione di errori. La terza alternativa è migliorare il modello. Per esempio, potremmo immettere conoscenza sulle parole, come le probabilità delle coppie di lettere. Di seguito considereremo questa opzione.

15.4.4 Migliorare il programma generativo per incorporare un modello di Markov

I linguaggi di programmazione probabilistica sono modulari in un modo che rende semplice esaminare i possibili miglioramenti del modello sottostante. La Figura 15.15 mostra il programma generativo per un modello perfezionato che genera le lettere in modo sequenziale invece che indipendente. Questo programma generativo utilizza un modello di Markov che estrae una lettera data la lettera precedente, con probabilità delle transizioni stimate sulla base di un elenco di vocaboli di riferimento.

La Figura 15.12 mostra dodici immagini prodotte da questo programma generativo. Notate che le sequenze di lettere appaiono assai più simili a parole naturali di quelle generate dal programma della Figura 15.11. La parte destra della Figura 15.14 mostra i risultati delle inferenze di questo modello di Markov applicato all'immagine con molto rumore. Le interpretazioni si avvicinano maggiormente alla traccia generatrice, sebbene permanga un po' di incertezza.

15.4.5 Inferenza nei programmi generativi

Come con i modelli OUPM, con i programmi generativi l'inferenza esatta è di solito proibitivamente dispendiosa se non impossibile. Per contro, si vede facilmente come eseguire un campionamento di rigetto: si esegue il programma tenendo solamente i tracciati compatibili

```
function GENERA-LETTERE-MARKOV( $\lambda$ ) returns un vettore di lettere
   $n \sim Poisson(\lambda)$ 
  lettere  $\leftarrow [ ]$ 
  prob_lettere  $\leftarrow$  MARKOV-INIZIALE( )
  for i = 1 to n do
    lettere[i]  $\sim Categorical(prob\_lettere)$ 
    prob_lettere  $\leftarrow$  TRANSIZIONE-MARKOV(lettere[i])
  return lettere
```

Figura 15.15 Programma generativo per un modello perfezionato di riconoscimento ottico dei caratteri, che genera le lettere secondo un modello a digrammi in cui le frequenze delle coppie di lettere sono stimate sulla base di un elenco di parole in linguaggio naturale.

con le evidenze e si contano le differenti risposte alla query trovate in questi tracciati. Anche la pesatura di verosimiglianza è semplice: per ogni tracciato generato, si individua il peso del tracciato moltiplicando tutte le probabilità dei valori osservati lungo il percorso.

La pesatura di verosimiglianza funziona bene solo quando i dati sono ragionevolmente plausibili secondo il modello. In casi più difficili, si sceglie di solito il metodo MCMC. L'applicazione di MCMC ai programmi probabilistici implica il campionamento e la modifica dei tracciati di esecuzione. Molte delle considerazioni fatte per i modelli OUPM sono valide anche in questo caso; inoltre, l'algoritmo deve essere cauto rispetto alle modifiche dei tracciati di esecuzione, come la modifica del risultato di un'istruzione if, che potrebbe invalidare il resto del tracciato.

Ulteriori miglioramenti dell'inferenza derivano da diversi ambiti. Alcuni miglioramenti possono produrre mutamenti fondamentali della classe di problemi trattabili con un dato PPL, anche in linea di principio; l'inferenza “sollevata”, descritta in precedenza per i modelli RPM, può avere questo effetto. In molti casi, il generico MCMC è troppo lento e sono necessarie proposte specializzate per consentire al processo di inferenza di operare rapidamente.

Recentemente, un importante campo di lavoro sui PPL è la ricerca di un modo per rendere semplice per gli utenti la definizione e l'utilizzo di simili proposte, in modo che l'efficienza dell'inferenza PPL raggiunga quella di algoritmi di inferenza personalizzati individuati per specifici modelli.

Molti approcci promettenti mirano a ridurre il costo dell'inferenza probabilistica. L'idea della compilazione descritta per le reti bayesiane nel Paragrafo 13.4.3 può essere applicata all'inferenza negli OUPM e nei PPL, e produce tipicamente velocizzazioni di due o tre ordini di grandezza. Ci sono state anche proposte di *hardware specializzato* per algoritmi come il passaggio di messaggi e MCMC. Per esempio, l'hardware Monte Carlo sfrutta rappresentazioni di probabilità a bassa precisione e utilizza massicciamente il parallelismo a grana fine per generare incrementi di 100–10.000 volte della velocità e dell'efficienza energetica.

distribuzione proposta adattiva

Anche i metodi basati sull'apprendimento possono portare a significativi incrementi della velocità. Per esempio, le **distribuzioni proposte adattive** possono gradualmente apprendere come generare proposte MCMC che abbiano ragionevoli possibilità di essere accettate e che siano ragionevolmente efficaci nell'esplorare il panorama di probabilità del modello per garantire un mixing rapido. È inoltre possibile addestrare i modelli di apprendimento deep (cfr. Capitolo 21 del Volume 2) per rappresentare le distribuzioni proposte per il campionamento dell'importanza, utilizzando dati sintetici generati dal modello sottostante.

In generale, ci si aspetta che ogni formalismo costruito su linguaggi di programmazione generici si imbatta nella barriera della calcolabilità, ed è ciò che avviene con i PPL. Se però assumiamo che il programma sottostante termini per tutti gli input e tutte le scelte casuali, il requisito di eseguire inferenze probabilistiche continua a far sì che il problema sia indecidibile? La risposta è sì, ma solo per i modelli computazionali con variabili casuali continue di precisione infinita. In quel caso, diventa possibile scrivere un modello probabilistico calcolabile in cui il problema della terminazione del programma sia codificato nell'inferenza. Viceversa, con numeri di precisione finita e con le distribuzioni di probabilità “regolari” tipicamente utilizzate nelle applicazioni reali, l'inferenza rimane decidibile.

15.5 Riepilogo

In questo capitolo abbiamo esaminato le rappresentazioni espressive per modelli probabilistici basati sia sulla logica sia su programmi.

- **I modelli relazionali di probabilità (RPM)** definiscono modelli probabilistici su mondi derivati dalla **semantica dei database** per linguaggi del primo ordine; sono appropriati quando tutti gli oggetti e le loro identità sono noti con certezza.

- Dato un modello RPM, gli oggetti di ciascun mondo possibile corrispondono ai simboli di costante del modello RPM e le variabili casuali di base sono tutte le possibili istanze dei simboli di predicato, con oggetti a sostituire ciascun argomento. Quindi, l'insieme dei mondi possibili è finito.
- I modelli RPM forniscono modelli molto succinti di mondi con grandi quantità di oggetti e possono gestire l'incertezza relazionale.
- **I modelli probabilistici a universo aperto** (OUPM) si fondano sulla semantica della logica del primo ordine, ammettendo nuovi tipi di incertezza come l'incertezza dell'identità e quella dell'esistenza.
- **I programmi generativi** sono rappresentazioni di modelli probabilistici (tra cui gli OUPM) come programmi eseguibili in un **linguaggio di programmazione probabilistico**, o **PPL**. Un programma generativo rappresenta una distribuzione sui **tracciati di esecuzione** del programma. I PPL offrono tipicamente capacità espressiva *universale* per i modelli di probabilità.

Note storiche e bibliografiche

Hailperin (1984) e Howson (2003) raccontano la lunga storia dei tentativi di collegare la probabilità e la logica, risalendo ai *Nouveaux Essais* di Leibniz del 1704. Solitamente, questi tentativi contemplavano probabilità direttamente legate a formule logiche. La prima trattazione rigorosa è la **logica probabilistica proposizionale** di Gaifman (Gaifman, 1964b). L'idea è che un'asserzione probabilistica $P(\phi) \geq p$ sia un vincolo sulla distribuzione sui mondi possibili, così come una formula di logica ordinaria è un vincolo sui mondi possibili stessi. Ogni distribuzione P che soddisfa il vincolo è un modello, nel senso logico standard, dell'asserzione probabilistica, e un'asserzione probabilistica ne implica un'altra quando i modelli della prima sono un sottoinsieme dei modelli della seconda.

Entro una simile logica è dimostrabile, per esempio, che $P(\alpha \wedge \beta) \leq P(\alpha \Rightarrow \beta)$. La soddisfattibilità di insiemi di asserzioni probabilistiche può essere determinata nel caso proposizionale mediante la programmazione lineare (Hailperin, 1984; Nilsson, 1986). Quindi, abbiamo una “logica probabilistica” nello stesso senso in cui abbiamo la “logica temporale”: un sistema logico specializzato per il ragionamento probabilistico.

Per applicare la logica probabilistica a compiti come la dimostrazione di teoremi interessanti della teoria della probabilità, era necessario un linguaggio maggiormente espressivo. Gaifman (1964a) propose una logica probabilistica del *primo ordine*, in cui i mondi possibili sono strutture di modelli del primo ordine e le probabilità sono associate a formule della logica del primo ordine (senza funzioni). Scott e Krauss

(1966) hanno esteso i risultati di Gaifman per consentire la nidificazione infinita dei quantificatori e insiemi infiniti di formule.

Nell'ambito dell'IA, la discendenza più diretta di queste idee appare nei **programmi logici probabilistici** (Lukasiewicz, 1998), in cui un intervallo di probabilità è associato a ogni clausola di Horn del primo ordine e le inferenze vengono effettuate risolvendo programmi lineari, come suggerito da Hailperin. Anche Halpern (1990) e Bacchus (1990) hanno sviluppato l'approccio di Gaifman, esplorando alcuni dei problemi fondamentali della rappresentazione della conoscenza dalla prospettiva della IA invece che da quella della teoria della probabilità e della logica matematica.

Anche nel sottocampo dei **database probabilistici** ci sono formule logiche associate a probabilità (Dalvi et al., 2009) ma in questo caso le probabilità sono associate direttamente alle tuple del database (in IA e in statistica, la probabilità è associata alle relazioni, mentre le osservazioni sono viste come evidenze inconfutabili). Sebbene i database probabilistici possano modellare dipendenze complesse, in pratica tali sistemi adottano spesso assunti di indipendenza globale sulle tuple.

L'associazione di probabilità alle formule rende molto difficile la definizione di modelli di probabilità completi e consistenti. Ogni diseguaglianza *vincola* il modello di probabilità sottostante a giacere in un semi-spazio nello spazio a molte dimensioni dei modelli di probabilità. Unire più asserzioni corrisponde a intersecare i vincoli. Fare in modo che l'intersezione ge-

neri un unico punto non è semplice. Infatti, il principale risultato in Gaifman (1964a) è la costruzione di un unico modello di probabilità che richiede 1) una probabilità per ogni possibile formula ground e 2) vincoli probabilistici per un numero infinito di formule quantificate esistenzialmente.

Una soluzione di questo problema consiste nello scrivere una teoria parziale e poi “completarla” scegliendo uno dei modelli canonici nell’insieme di quelli ammissibili. Nilsson (1986) propone di scegliere il modello di *massima entropia* coerente con i vincoli specificati. Paskin (2002) ha sviluppato una “logica probabilistica di massima entropia” con vincoli espressi come pesi (probabilità relative) associati a clausole del primo ordine. Simili modelli sono spesso chiamati **reti logiche di Markov** o MLN, da *Markov logic network* (Richardson e Domingos, 2006) e sono diventati una tecnica popolare per applicazioni che coinvolgono dati relazionali. Gli approcci di massima entropia, compresi i MLN, possono in alcuni casi produrre risultati poco intuitivi (Milch, 2006; Jain *et al.*, 2007, 2010).

A partire dai primi anni 1990, i ricercatori che lavorano su applicazioni complesse hanno notato i limiti espressivi delle reti bayesiane e hanno sviluppato vari linguaggi per scrivere degli “schemi” o “template” con variabili logiche, dai quali sia possibile costruire automaticamente reti di grandi dimensioni per ogni istanza del problema (Breese, 1992; Wellman *et al.*, 1992). Il più importante di questi linguaggi era BUGS (Bayesian inference Using Gibbs Sampling) (Gilks *et al.*, 1994; Lunn *et al.*, 2013), che univa le reti bayesiane alla notazione delle **variabili casuali indicizzate**, comune in statistica (in BUGS, una variabile casuale indicizzata ha forma $X[i]$, dove i appartiene a un intervallo di interi definito).

Questi linguaggi a universo chiuso hanno ereditato la proprietà chiave delle reti bayesiane: ogni base di conoscenza ben formata definisce un unico e coerente modello di probabilità. Altri linguaggi a universo chiuso sfruttavano le capacità di rappresentazione e di inferenza della programmazione logica (Poole, 1993; Sato e Kameya, 1997; Kersting *et al.*, 2000) e delle reti semantiche (Koller e Pfeffer, 1998; Pfeffer, 2000).

La ricerca sui modelli probabilistici a universo aperto ha diverse origini. In statistica, il problema del **record linkage** si presenta quando i record di dati non contengono identificatori univoci standard: per esempio varie citazioni bibliografiche di questo libro potrebbero indicare il primo autore come “Stuart J. Russell” o come “S. Russell” oppure come “Stewart Russel”. Ci sono anche altri autori di nome “S. Russell”.

Esistono centinaia di aziende il cui lavoro è unicamente risolvere problemi di record linkage in ambito finanziario, medico, anagrafico e così via. L’analisi probabilistica risale al lavoro di Dunn (1946); il modello Fellegi–Sunter (1969), che è essenzialmente un algoritmo di Bayes ingenuo (*naive Bayes*) applicato all’individuazione di corrispondenze, è ancora dominante nelle applicazioni pratiche. L’incertezza dell’identità è presa in considerazione anche nel tracciamento multitarget (Sittler, 1964).

Fino agli anni 1990, l’assunto operativo in IA era che i sensori possano fornire formule logiche con identificatori univoci per gli oggetti, come nel caso di Shakey. Nell’area della comprensione del linguaggio naturale, Charniak e Goldman (1992) hanno proposto un’analisi probabilistica della coreferenza, cioè dei casi in cui due espressioni linguistiche (per esempio, “Obama” e “il presidente degli Stati Uniti”) possono riferirsi alla stessa entità. Huang e Russell (1998) e Pasula *et al.* (1999) hanno sviluppato un’analisi bayesiana dell’incertezza dell’identità per il monitoraggio del traffico. Pasula *et al.* (2003) hanno sviluppato un modello generativo complesso per autori, pubblicazioni e stringhe bibliografiche, che coinvolge sia l’incertezza relazionale sia quella dell’identità, e che ha dimostrato un’elevata accuratezza nell’estrazione delle informazioni bibliografiche.

Il primo linguaggio formale per i modelli probabilistici a universo aperto è stato BLOG (Milch *et al.*, 2005; Milch, 2006), che aveva un motore di inferenza MCMC generico (molto lento). Laskey (2008) descrive un altro linguaggio di modellazione a universo aperto chiamato **reti bayesiane multientità**. Il sistema di monitoraggio sismologico globale NET-VISA descritto nel testo si deve ad Arora *et al.* (2013). Il sistema di classificazione Elo è stato sviluppato nel 1959 da Arpad Elo (1978) ma è sostanzialmente uguale al modello Case V di Thurstone (Thurstone, 1927). Il modello Microsoft TrueSkill (Herbrich *et al.*, 2007; Minka *et al.*, 2018) è basato sulla versione bayesiana di Elo proposta da Mark Glickman (1999) e oggi viene eseguito sul PPL infer.NET.

L’associazione di dati per il tracciamento di più oggetti è stata descritta per la prima volta in ambito probabilistico da Sittler (1964). Il primo algoritmo pratico per problemi su grande scala era l’algoritmo “multiple hypothesis tracker” o MHT (Reid, 1979). Articoli importanti sono raccolti in Bar-Shalom e Fortmann (1988) e in Bar-Shalom (1992). Lo sviluppo di un algoritmo MCMC per l’associazione dei dati si deve a Pasula *et al.* (1999), che lo hanno applicato a problemi di

monitoraggio del traffico. Oh *et al.* (2009) hanno proposto un'analisi formale e confronti sperimentali con altri metodi. Schulz *et al.* (2003) descrivono un metodo di associazione dei dati basato sul particle filtering.

Ingemar Cox ha analizzato la complessità dell'associazione dei dati (Cox, 1993; Cox e Hingorani, 1994) e ha portato l'argomento all'attenzione della comunità di visione artificiale. Ha inoltre notato l'applicabilità dell'algoritmo ungherese, che ha tempo polinomiale, al problema di trovare gli assegnamenti più probabili, a lungo considerato un problema non trattabile da chi si occupa di tracciamento. L'algoritmo era stato pubblicato da Kuhn (1955), sulla base di traduzioni di studi pubblicati nel 1931 da due matematici ungheresi, Dénes König e Jenö Egerváry. Il teorema base era stato tuttavia ricavato in precedenza e si trova in un manoscritto latino non pubblicato del famoso matematico Carl Gustav Jacobi (1804–1851).

L'idea che i programmi probabilistici possano rappresentare anche modelli probabilistici complessi si deve a Koller *et al.* (1997). Il primo PPL funzionale è stato IBAL di Avi Pfeffer (2001, 2007), basato su un semplice linguaggio funzionale. BLOG può essere considerato un PPL dichiarativo. Il collegamento tra PPL dichiarativi e funzionali è stato esplorato da McAllester *et al.* (2008). CHURCH (Goodman *et al.*, 2008), un PPL costruito sul linguaggio Scheme, ha introdotto l'idea di appoggiarsi a un linguaggio di programmazione esistente. CHURCH ha introdotto anche il primo algoritmo di inferenza MCMC per modelli con funzioni casuali di ordine superiore e ha suscitato l'interesse della comunità delle scienze cognitive rispetto alla possibilità di modellare forme complesse di apprendimento umano (Lake *et al.*, 2015). I PPL sono inoltre connessi in modi interessanti alla teoria della calcolabilità (Ackerman *et al.*, 2013) e alla ricerca sui linguaggi di programmazione.

Negli anni 2010, sono emerse decine di PPL basati su una vasta gamma di linguaggi di programmazione. Figaro, basato sul linguaggio Scala, è stato utilizzato per una grande varietà di applicazioni (Pfeffer, 2016). Gen (Cusumano-Towner *et al.*, 2019), basato su Julia e TensorFlow, è stato utilizzato per la percezione artificiale in tempo reale e per l'apprendimento bayesiano applicato all'analisi di serie di dati storici. Esempi di PPL costruiti su piattaforme di apprendimento deep sono Pyro (Bingham *et al.*, 2019) (basato su PyTorch) ed Edward (Tran *et al.*, 2017) (basato su TensorFlow).

Ci sono stati dei tentativi per rendere la programmazione probabilistica più accessibile, per esempio, a chi utilizza i database e i fogli elettronici. Tabular

(Gordon *et al.*, 2014) fornisce un linguaggio per schemi relazionali come quelli usati nei fogli di calcolo, basato su infer.NET. BayesDB (Saad e Mansinghka, 2017) consente agli utenti di combinare e interrogare programmi probabilistici utilizzando un linguaggio simile a SQL.

L'inferenza nei programmi probabilistici ha generalmente fatto affidamento su metodi approssimati, perché gli algoritmi esatti non sono compatibili con la scala dei modelli che possono essere rappresentati dai PPL. I linguaggi a universo chiuso come BUGS, LIBBi (Murray, 2013) e STAN (Carpenter *et al.*, 2017) generalmente costruiscono l'intera rete bayesiana equivalente per poi eseguire l'inferenza su di essa: campionamento di Gibbs nel caso di BUGS, Monte Carlo sequenziale nel caso di LIBBi e Monte Carlo hamiltoniano in quello di STAN. I programmi scritti in questi linguaggi possono essere letti come istruzioni per la costruzione della rete bayesiana ground. Breese (1992) mostra come generare solamente la porzione rilevante della rete, date la query e le evidenze.

Lavorare con una rete bayesiana ground significa che i mondi possibili visitati da MCMC sono rappresentati da vettori di valori per le variabili della rete. L'idea di campionare direttamente mondi possibili del primo ordine si deve a Russell (1999). Nel linguaggio FACTORIE (McCallum *et al.*, 2009), i mondi possibili nel processo MCMC sono rappresentati all'interno di un database relazionale standard. Gli stessi due articoli accademici propongono la rivalutazione incrementale della query come metodo per evitare la valutazione completa in ogni mondo possibile.

I metodi di inferenza basati sul grounding sono analoghi ai primi metodi di proposizionalizzazione per l'inferenza logica del primo ordine (Davis e Putnam, 1960). Per l'inferenza logica, sia i dimostratori di teoremi basati sulla risoluzione sia i sistemi di programmazione logica poggiano sul **lifting** (o "sollevamento", Paragrafo 9.2), per evitare di creare istanze delle variabili logiche non necessarie.

Pfeffer *et al.* (1999) hanno introdotto un algoritmo di eliminazione di variabili che conserva tutti i fattori calcolati per riutilizzarli in calcoli successivi che coinvolgono le stesse relazioni ma oggetti differenti, ottenendo così una parte dei miglioramenti in termini di calcolo consentiti dal "sollevamento". Il primo vero algoritmo di inferenza probabilistico "sollevato" era una forma di eliminazione di variabili; è stato descritto da Poole (2003) e successivamente migliorato da de Salvo Braz *et al.* (2007). Ulteriori progressi, riguardanti anche casi in cui alcune probabilità aggregate pos-

sono essere calcolate in forma chiusa, sono descritti da Milch *et al.* (2008) e da Kisynski e Poole (2009). Oggi abbiamo una conoscenza soddisfacente della complessità del lifting e dei casi in cui è possibile (Gribkoff *et al.*, 2014; Kazemi *et al.*, 2017).

I metodi per velocizzare l’inferenza sono di vario genere, come mostrato nel capitolo. Diversi progetti hanno esplorato algoritmi più sofisticati, combinati con tecniche di compilazione e/o apprendimento di proposte. LIBBI (Murray, 2013) ha introdotto l’inferenza basata sul campionamento di Gibbs per i programmi probabilistici; uno dei primi compilatori per inferenze, con supporto GPU per seguire SMC in parallelo; e l’utilizzo del linguaggio di modellazione per definire proposte MCMC personalizzate. La compilazione dell’inferenza probabilistica è stata studiata anche da Wingate *et al.* (2011), Paige e Wood (2014), Wu *et al.* (2016a). Claret *et al.* (2013), Hur *et al.* (2014) e Cusumano-Towner *et al.* (2019) mostrano metodi di analisi statistica per trasformare i programmi probabilistici in forme più efficienti. PICTURE (Kulkarni *et al.*, 2015) è il primo PPL che consente agli utenti di applicare l’apprendimento basato sulle esecuzioni in

avanti del programma generativo per generare proposte bottom-up in modo rapido. Le *et al.* (2017) descrivono l’impiego delle tecniche di apprendimento deep per eseguire il campionamento di importanza in modo efficiente in un PPL. In pratica, gli algoritmi di inferenza per i modelli probabilistici complessi spesso ricorrono a un mix di tecniche per diversi sottoinsiemi delle variabili del modello. Mansinghka *et al.* (2013) hanno sottolineato l’idea dei programmi che applicano differenti tattiche di inferenza a sottoinsiemi di variabili scelti durante l’esecuzione dell’inferenza.

La raccolta curata da Getoor e Taskar (2007) comprende molti importanti articoli sui modelli probabilistici del primo ordine e sul loro utilizzo nell’apprendimento automatico. Articoli sulla programmazione probabilistica appaiono in tutte le principali conferenze sull’apprendimento automatico e sul ragionamento probabilistico, tra cui NeurIPS, ICML, UAI e AI-STATS. Le conferenze NeurIPS e POPL (Principles of Programming Languages) sono accompagnate regolarmente da workshop sui PPL e nel 2018 si è tenuta la prima edizione della International Conference on Probabilistic Programming.

Decisioni semplici

- 16.1 Combinare credenze e desideri in condizioni di incertezza
- 16.2 Le basi della teoria dell'utilità
- 16.3 Funzioni di utilità
- 16.4 Funzioni di utilità multiattributo
- 16.5 Reti di decisione
- 16.6 Il valore dell'informazione
- 16.7 Preferenze ignote
- 16.8 Riepilogo
 - Note storiche e bibliografiche

In cui vediamo come un agente dovrebbe prendere decisioni in modo da ottenere ciò che vuole in un mondo incerto, almeno per quanto possibile e in media.

In questo capitolo esaminiamo come la teoria dell'utilità si combina con la teoria della probabilità per ottenere un agente basato sulla teoria delle decisioni, che può scegliere razionalmente l'azione da intraprendere in base a ciò che crede e desidera. Un simile agente potrà operare in contesti in cui incertezza e conflitti tra obiettivi contrastanti non consentono a un agente puramente logico di arrivare ad alcuna decisione. Un agente basato su obiettivi può solo dividere in modo binario gli stati in buoni (obiettivo) e cattivi (non obiettivo), mentre un agente basato sulla teoria delle decisioni assegna agli stati un intervallo continuo di valori, perciò è in grado di scegliere più facilmente uno stato migliore anche quando non è disponibile alcuno stato migliore in assoluto.

Il Paragrafo 16.1 introduce il principio base della teoria delle decisioni: massimizzare l'utilità attesa. Il Paragrafo 16.2 mostra che il comportamento di un agente razionale può essere modellato come massimizzazione di una funzione di utilità. Il Paragrafo 16.3 discute in maggior dettaglio la natura delle funzioni di utilità, con particolare attenzione verso quantità come il denaro. Il Paragrafo 16.4 mostra come gestire funzioni di utilità che dipendono da diverse quantità. Nel Paragrafo 16.5 descriviamo l'implementazione dei sistemi il cui scopo è prendere decisioni: in particolare presentiamo un formalismo chiamato **rete di decisione** (noto anche come **diagramma di influenza**) che estende le reti bayesiane incorporandovi azioni e utilità. Nel Paragrafo 16.6 mostriamo come un agente basato sulla teoria delle decisioni possa calcolare il valore dell'acquisizione di nuove informazioni per migliorare le sue decisioni.

Nei Paragrafi da 16.1 a 16.6 ipotizziamo che l'agente operi con una funzione di utilità data e conosciuta, mentre nel Paragrafo 16.7 rilassiamo questa ipotesi. Discuteremo le conseguenze dell'incertezza sulle preferenze per quanto riguarda la macchina; la conseguenza più importante è il fatto che la macchina si affida agli esseri umani.

16.1 Combinare credenze e desideri in condizioni di incertezza

Iniziamo con un agente che, come tutti gli agenti, deve prendere una decisione. Sono disponibili alcune azioni a . Potrebbe esserci dell'incertezza sullo stato corrente, perciò ipotizzeremo che l'agente assegna una probabilità $P(s)$ a ogni possibile stato corrente s . Potrebbe esserci incertezza anche sugli esiti dell'azione; il modello di transizione è dato da $P(s' | s, a)$, la probabilità che l'azione a nello stato s raggiunga lo stato s' . Poiché siamo interessati principalmente all'esito s' , utilizzeremo la notazione abbreviata $P(\text{RISULTATO}(a) = s')$, la probabilità di raggiungere s' eseguendo l'azione a nello stato corrente, qualunque sia. Abbiamo la seguente relazione:

$$P(\text{RISULTATO}(a) = s') = \sum_s P(s) P(s' | s, a).$$

**funzione di utilità
utilità attesa**

La teoria delle decisione, nella forma più semplice, si occupa di scegliere tra azioni in base al grado di desiderabilità dei loro esiti *immediati*; ovvero, si ipotizza che l'ambiente sia episodico nel senso definito nel Paragrafo 2.3.2 (questa ipotesi viene rilassata nel Capitolo 17). Le preferenze dell'agente sono catturate da una **funzione di utilità**, $U(s)$, che assegna un singolo numero per esprimere la desiderabilità di uno stato. L'**utilità attesa** di un'azione data l'evidenza, $EU(a)$, è semplicemente il valore di utilità medio degli esiti, pesato in base alla probabilità che l'esito si verifichi:

$$EU(a) = \sum_{s'} P(\text{RISULTATO}(a) = s') U(s'). \quad (16.1)$$

Il principio della **massima utilità attesa** (**MEU**, *maximum expected utility*) afferma che un agente razionale dovrebbe scegliere l'azione che massimizza l'utilità attesa dell'agente:

$$\text{azione} = \underset{a}{\operatorname{argmax}} EU(a).$$

In un certo senso, il principio della massima utilità attesa può essere visto come una prescrizione per un comportamento intelligente. Tutto ciò che un agente intelligente deve fare è calcolare le varie quantità, massimizzare l'utilità sulle sue azioni ed ecco tutto. Ma questo non significa che il problema dell'IA sia *risolto* da tale definizione!

Il principio MEU *formalizza* il concetto generale per cui un agente intelligente dovrebbe “fare la cosa giusta”, ma non traduce in *operatività* tale concetto. Per stimare la distribuzione di probabilità $P(s)$ sui possibili stati del mondo, che si incorpora in $P(\text{RISULTATO}(a) = s')$, servono percezione, apprendimento, rappresentazione della conoscenza e inferenza. Per calcolare $P(\text{RISULTATO}(a) = s')$ è necessario possedere un modello causale del mondo. Potrebbero esserci molte azioni da considerare, e il calcolo delle utilità degli esiti $U(s')$ potrebbe richiedere a sua volta ulteriori ricerche o pianificazioni, perché un agente potrebbe non sapere quanto è buono uno stato finché non sa dove può arrivare a partire da quello stato. Per riassumere, la teoria delle decisioni non è la panacea che risolve il problema dell'IA, ma fornisce l'inizio di un'infrastruttura sufficientemente generale per definire il problema dell'IA.

Il principio MEU è chiaramente collegato all'idea delle misure di prestazione introdotte nel Capitolo 2. L'idea base è semplice: consideriamo gli ambienti che potrebbero fornire a un agente una particolare storia percettiva e consideriamo i diversi agenti che potremmo progettare. *Se un agente agisce in modo da massimizzare una funzione di utilità che riflette correttamente la misura di prestazione, allora otterrà la migliore prestazione possibile, calcolata sulla media di tutti gli ambienti in cui potrà operare.* Questa è la giustificazione fondamentale dello stesso principio MEU. Sebbene la frase possa sembrare tautologica, racchiude di fatto un passaggio molto importante da una misura di prestazione esterna a una funzione



di utilità interna. La misura di prestazione fornisce un punteggio per una storia – una sequenza di stati. Viene quindi applicata retrospettivamente dopo che un agente completa una sequenza di azioni. La funzione di utilità si applica allo stato immediatamente successivo e può dunque essere usata per guidare le azioni passo per passo.

16.2 Le basi della teoria dell'utilità

Intuitivamente, il principio della massima utilità attesa (MEU) sembra un modo ragionevole di prendere decisioni, ma non è affatto ovvio che sia anche l'*unico* modo razionale. Dopo tutto, perché dovrebbe essere così speciale massimizzare l'utilità *media*? Che cosa non va in un agente che massimizza la somma pesata dei cubi di tutte le possibili utilità, o cerca di minimizzare la perdita peggiore? E ancora, un agente potrebbe agire razionalmente limitandosi a esprimere una preferenza tra gli stati, anziché assegnare loro dei valori numerici? Infine, perché dovrebbe esistere una funzione di utilità con le proprietà richieste? Lo vedremo.

16.2.1 Vincoli su preferenze razionali

A tutte queste domande si può rispondere scrivendo alcuni vincoli sulle preferenze che un agente razionale dovrebbe rispettare, e mostrando poi che da tali vincoli si può derivare il principio MEU. Per descrivere le preferenze useremo la seguente notazione:

- $A > B$ l'agente preferisce A a B .
- $A \sim B$ l'agente è indifferente tra A e B .
- $A \gtrsim B$ l'agente preferisce A a B o è indifferente.

A questo punto sorge l'ovvia questione: che cosa sono A e B ? Potrebbero essere stati del mondo, ma più spesso c'è incertezza su cosa siano davvero. Per esempio, un passeggero di un aereo a cui viene offerta la scelta tra “piatto di pasta o pollo” non sa quali prelibatezze si nascondano sotto il coperchio di alluminio.¹ La pasta potrebbe essere deliziosa o magari sur-gelata, il pollo saporito o stracotto. Possiamo pensare all'insieme degli esiti per ogni azione come a una **lotteria** – in cui ogni azione è un biglietto. Una lotteria L con i possibili esiti S_1, \dots, S_n , che si verificano con probabilità p_1, \dots, p_n , si scrive:

$$L = [p_1, S_1; p_2, S_2; \dots; p_n, S_n].$$

In generale, ogni esito S_i di una lotteria può essere uno stato atomico o un'altra lotteria. Il problema principale della teoria dell'utilità è comprendere il collegamento tra le preferenze relative a lotterie complesse e quelle relative agli stati sottostanti queste lotterie. Per affrontare questo problema elenchiamo sei vincoli che richiediamo a ogni relazione di preferenza ragionevole.

lotteria

- **Ordinabilità:** date due lotterie, un agente razionale deve preferire l'una o l'altra o considerarle ugualmente preferibili. In altre parole, l'agente non può rifiutarsi di decidere. Come si è osservato nel Paragrafo 12.2.3, rifiutare di scommettere è come cercare di impedire lo scorrere del tempo.

ordinabilità

Esattamente uno tra $(A > B)$, $(B > A)$ o $(A \sim B)$ vale.

- **Transitività:** date tre lotterie, se un agente preferisce A rispetto a B e B rispetto a C , allora deve preferire A rispetto a C .

transitività

$$(A > B) \wedge (B > C) \Rightarrow (A > C).$$

¹ Ci scusiamo con i lettori che fanno viaggi in aereo dove non vengono offerti pasti a bordo.

continuità

- **Continuità:** se una lotteria B ha preferenza compresa tra A e C , deve esistere una qualche probabilità p per cui l'agente razionale sarà indifferente tra ottenere B con certezza e partecipare alla lotteria che fornisce A con probabilità p e C con probabilità $1-p$.

$$A > B > C \Rightarrow \exists p [p, A; 1-p, C] \sim B.$$

sostituibilità

- **Sostituibilità:** se un agente è indifferente tra due lotterie A e B , allora l'agente sarà indifferente anche tra due lotterie più complesse che sono identiche se non per il fatto che in una di esse B è sostituito da A . Questo vale indipendentemente dalle probabilità e dagli altri esiti nelle lotterie.

$$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C].$$

Questo vale anche se sostituiamo $>$ al posto di \sim in questo assioma.

monotonicità

- **Monotonicità:** supponiamo che due lotterie abbiano gli stessi esiti possibili: A e B . Se un agente preferisce A rispetto a B , allora l'agente deve preferire la lotteria che ha una probabilità più alta per A (e viceversa).

$$A > B \Rightarrow (p > q \Leftrightarrow [p, A; 1-p, B] > [q, A; 1-q, B]).$$

scomponibilità

- **Scomponibilità:** lotterie complesse possono essere ridotte a lotterie più semplici usando le leggi della probabilità. Questa regola è stata chiamata “no fun in gambling” (*non c'è divertimento nel gioco d'azzardo*), perché, come si vede nella Figura 16.1(b), riunisce due lotterie consecutive in una singola lotteria equivalente.²

$$[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C].$$

Questi vincoli sono noti come assiomi della teoria dell'utilità. Ogni assioma può essere motivato mostrando che un agente che lo violi presenta un comportamento evidentemente ir-

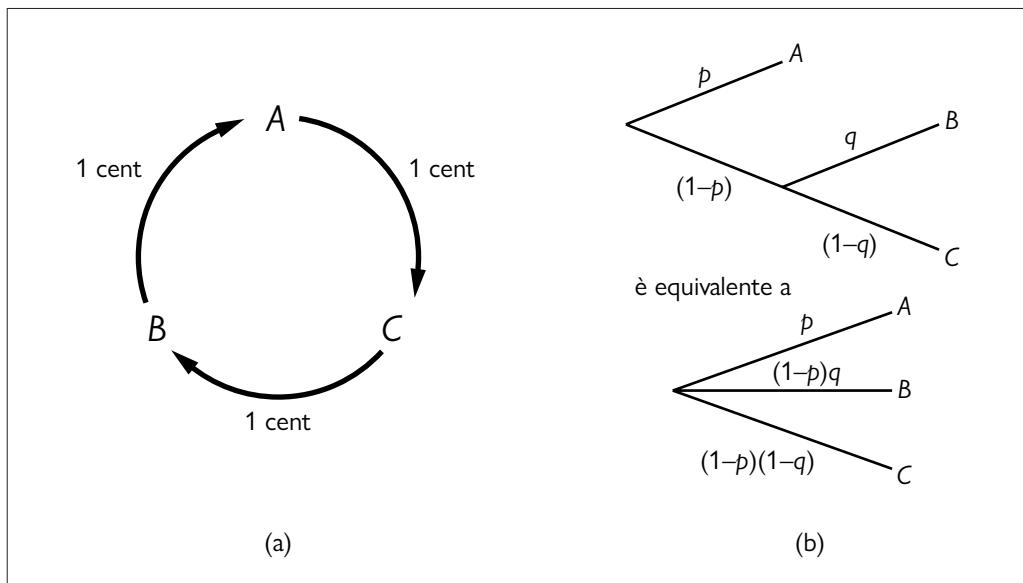


Figura 16.1 (a) Preferenze non transitive $A > B > C > A$ possono portare a un comportamento irrazionale: un ciclo di scambi, ognuno dei quali costa un centesimo. (b) L'assioma di scomponibilità.

² Possiamo rappresentare il divertimento intrinseco nel gioco d'azzardo codificando l'evento di scommettere nella descrizione degli stati; per esempio, “ho 10 euro e ho giocato” potrebbe essere preferibile rispetto a “ho 10 euro e non ho giocato”.

razionale in alcune situazioni. Per esempio, possiamo motivare la transitività facendo vedere che un agente con preferenze non transitive ci consegna tutto il suo denaro. Supponiamo che l'agente abbia le preferenze non transitive $A > B > C > A$, dove A, B e C sono beni a libero scambio. Se l'agente attualmente ha il bene A , potremmo fare un'offerta per scambiare C per A più un centesimo. L'agente preferisce C , quindi sarà disponibile allo scambio. Potremmo poi offrire B in cambio di C , ricavando un altro centesimo, e alla fine scambiare A per B . Torniamo così al punto da dove siamo partiti, con la differenza che l'agente ci ha dato tre centesimi (Figura 16.1(a)). Possiamo continuare a percorrere il ciclo finché l'agente esaurirà tutto il suo denaro. Chiaramente in questo caso l'agente ha avuto un comportamento irrazionale.

16.2.2 Le preferenze razionali conducono all'utilità

Notate che gli assiomi della teoria dell'utilità in realtà parlano di preferenze e non dicono nulla su funzioni di utilità. Tuttavia, dagli assiomi di utilità possiamo derivare alcune conseguenze (per le dimostrazioni cfr. von Neumann e Morgenstern, 1944).

- **Esistenza della funzione di utilità.** Se le preferenze di un agente obbediscono agli assiomi dell'utilità, allora esiste una funzione U tale che $U(A) > U(B)$ se e solo se A è preferito a B , e $U(A) = U(B)$ se e solo se l'agente è indifferente tra A e B . In altri termini:

$$U(A) > U(B) \Leftrightarrow A > B \quad \text{e} \quad U(A) = U(B) \Leftrightarrow A \sim B.$$

- **Utilità attesa di una lotteria.** L'utilità di una lotteria è la somma delle probabilità di ogni esito moltiplicata per l'utilità di tale esito.

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i).$$

In altre parole, una volta che sono state specificate le probabilità e le utilità di tutti i possibili stati risultanti, l'utilità della lotteria che coinvolge quegli stati è completamente determinata. Dato che l'esito di un'azione non deterministica è una lotteria, ne deriva che un agente può comportarsi razionalmente, cioè in modo coerente con le sue preferenze, soltanto scegliendo un'azione che massimizzi l'utilità attesa in base all'Equazione (16.1).

I teoremi precedenti affermano che (ipotizzando i vincoli sulla razionalità delle preferenze) una funzione di utilità *esiste* per qualsiasi agente razionale, non che la funzione di utilità è *unica*. È facile vedere, infatti, che il comportamento di un agente non cambierebbe se la sua funzione di utilità $U(S)$ fosse trasformata secondo la seguente equazione:

$$U'(S) = aU(S) + b, \tag{16.2}$$

dove a e b sono costanti e $a > 0$; si tratta di una trasformazione affine positiva.³ Questo fatto è stato osservato nel Capitolo 5 (Paragrafo 5.5.1) per i giochi stocastici a due giocatori; in questo caso vediamo che vale per ogni tipo di scenario decisionale.

Come nei giochi, in un ambiente deterministico un agente necessita soltanto di una graduatoria di preferenza sugli stati, i numeri non contano. Parliamo di **funzione valore** o **funzione di utilità ordinale**.

È importante ricordare che l'esistenza di una funzione di utilità che descrive le preferenze di un agente non significa necessariamente che l'agente stia massimizzando *esplicitamente* tale funzione nelle sue deliberazioni. Come abbiamo mostrato nel Capitolo 2, un comportamento razionale può essere generato in molti modi diversi. Un agente razionale potrebbe

funzione valore
funzione di utilità
ordinale

³ In questo senso le utilità assomigliano alle temperature, infatti una temperatura espressa in gradi Fahrenheit è pari a 1,8 volte la stessa temperatura espressa in gradi Celsius più 32, ma la conversione da un sistema all'altro non comporta un riscaldamento o un raffreddamento.

essere implementato con una ricerca in tabella (se il numero dei possibili stati è sufficientemente piccolo).

Osservando il comportamento di un agente razionale, un osservatore può apprendere la funzione di utilità che rappresenta ciò che l'agente sta effettivamente cercando di ottenere (anche se l'agente non lo sa). Torneremo su questo punto nel Paragrafo 16.7.

16.3 Funzioni di utilità

Le funzioni di utilità fanno corrispondere le lotterie a numeri reali. Sappiamo che devono rispettare gli assiomi di ordinabilità, transitività, continuità, sostituibilità, monotonicità e scomponibilità. E questo è tutto ciò che possiamo dire su di esse? A rigore di termini è così: un agente è libero di avere le preferenze che vuole. Potrebbe darsi per esempio che preferisca avere un numero primo di euro sul conto bancario: in tal caso, qualora avesse 16 euro ne darebbe via 3. Potrebbe sembrare strano, ma non possiamo dire che sia irrazionale. Un agente potrebbe preferire una FIAT Panda ammaccata del 1983 a una Mercedes nuova fiammante. Potrebbe preferire numeri primi di euro quando possiede la Panda, ma il numero più grande possibile quando invece ha la Mercedes. Fortunatamente, le preferenze degli agenti reali solitamente sono più sistematiche e quindi più facili da gestire.

16.3.1 Valutazione e scale di utilità

elicitazione delle preferenze

Se vogliamo costruire un sistema di teoria delle decisioni che aiuti un essere umano a prendere decisioni o che agisca per conto della persona, dobbiamo prima determinare qual è la funzione di utilità dell'essere umano. Questo processo, spesso chiamato **elicitazione delle preferenze**, comporta il fatto di presentare delle scelte all'essere umano e usare le preferenze osservate per ricavarne la funzione di utilità.

utilità normalizzate

L'Equazione (16.2) indica che non esiste una scala assoluta per le utilità, ma è comunque utile stabilire *qualche* scala su cui sia possibile registrare e confrontare le utilità per qualsiasi problema specifico. Per stabilire una scala si possono fissare le utilità di due esiti qualsiasi, proprio come si fa con la scala di temperatura, per cui fissiamo il punto di congelamento e di ebollizione dell'acqua. Generalmente fissiamo l'utilità del "miglior premio possibile" come $U(S) = u_{\top}$ e della "peggiore catastrofe possibile" come $U(S) = u_{\perp}$ (devono essere entrambi finiti). Le **utilità normalizzate** usano una scala con $u_{\perp} = 0$ e $u_{\top} = 1$. Con tale scala, un tifoso dell'Italia potrebbe assegnare un'utilità di 1 a una vittoria dell'Italia nella Coppa del Mondo e di 0 in caso di mancata qualificazione.

lotteria standard

Data una scala di utilità tra u_{\top} e u_{\perp} , possiamo valutare l'utilità di qualsiasi premio particolare S chiedendo all'agente di scegliere tra S e una **lotteria standard** $[p, u_{\top}; (1-p), u_{\perp}]$. La probabilità p viene modificata finché l'agente è indifferente tra S e la lotteria standard. Ipotizzando utilità normalizzate, l'utilità di S è data da p . Una volta fatto ciò per ogni premio, si determinano le utilità per tutte le lotterie che coinvolgono tali premi. Per esempio, supponiamo di voler conoscere quanto valutino i tifosi dell'Italia la conquista delle semifinali per poi uscire dal torneo. Confrontiamo tale esito con una lotteria standard con probabilità p di vincere il trofeo e $1 - p$ di mancare la qualificazione. Se c'è indifferenza con $p = 0,3$, allora 0,3 è il valore di raggiungere le semifinali e poi perdere.

Nei problemi decisionali in campo medico, dei trasporti, dell'ambiente e altri, entrano in gioco le vite delle persone (sì, esistono cose più importanti delle sorti della nazionale alla Coppa del Mondo). In tali casi, u_{\perp} è il valore assegnato alla morte immediata (o, nei casi peggiori, all'eventualità di molte morti). *Benché nessuno si senta a suo agio nell'assegnare un valore alla vita umana, in realtà si fanno continuamente compromessi su questioni di vita e di morte.* Gli aeroplani sono sottoposti a una revisione completa a intervalli periodici, non



dopo ogni volo. Le automobili vengono fabbricate in modo da bilanciare i costi e i tassi di sopravvivenza in caso di incidente. Tolleriamo un livello di inquinamento atmosferico che uccide quattro milioni di persone all'anno.

Paradossalmente, il rifiuto di assegnare un valore monetario alla vita umana può significare che la vita è *sottovalutata*. Ross Shachter descrive un ente pubblico che commissionò uno studio per la rimozione dell'amianto dalle scuole. Gli analisti decisionali che svolsero lo studio ipotizzarono un particolare valore monetario per la vita di un ragazzo in età scolare e sostennero che la scelta razionale, sotto tale ipotesi, fosse quella di rimuovere l'amianto. L'ente pubblico, moralmente indignato dall'idea di attribuire un valore alla vita, respinse la relazione e decise poi di non rimuovere l'amianto, finendo così per attribuire – in modo implicito – alla vita di un bambino un valore inferiore a quello che era stato assegnato dagli analisti incaricati dello studio.

Attualmente, diversi enti del governo USA, tra cui l'EPA (*Environmental Protection Agency*), la FDA (*Food and Drug Administration*) e il dipartimento dei trasporti, utilizzano il **valore di una vita statistica** per determinare i costi e i benefici delle norme e degli interventi. Nel 2019 tale valore è dell'ordine di 10 milioni di dollari.

valore di una vita statistica

Sono stati anche compiuti dei tentativi di determinare il valore assegnato dalle persone alla propria stessa vita: una “valuta” usata comunemente nell’analisi medica e della sicurezza è il **micromort**, una possibilità su un milione di morire. Se si chiede alle persone quanto pagherebbero per evitare un rischio, per esempio per evitare di giocare alla roulette russa con una pistola dotata di un milione di proiettili, risponderebbero indicando numeri molto grandi, magari decine di migliaia di euro, ma il loro comportamento effettivo potrebbe riflettere un valore monetario molto inferiore attribuito a un micromort.

micromort

Per esempio, nel Regno Unito guidando l’auto per 230 miglia si corre il rischio di un micromort. Nell’arco di vita di un’auto, diciamo di circa 92.000 miglia, ci sono quindi 400 micromort. Le persone sembrano disponibili a pagare circa 12.000 euro in più per un’auto più sicura che dimezzi il rischio di morte. Perciò, la loro azione di acquisto di auto afferma che attribuiscono al micromort il valore di 60 euro. Diversi studi hanno confermato una cifra in questo intorno per molti tipi di individuo e di rischio. Tuttavia, gli enti pubblici quali il dipartimento dei trasporti USA generalmente fissano una cifra più bassa: arrivano a spendere al più 6 euro di riparazioni stradali per vita attesa risparmiata. Naturalmente questi calcoli valgono solo per rischi piccoli. La maggior parte delle persone non ha nessuna intenzione di uccidersi, nemmeno per 60 milioni di euro.

QALY

Un’altra misura è il **QALY**, acronimo di *quality-adjusted life year* (letteralmente “anno di vita corretto per la qualità”). I pazienti sono disponibili ad accettare un’aspettativa di vita più breve per evitare una disabilità. Per esempio, i pazienti con problemi ai reni in media sono indifferenti tra vivere due anni in dialisi e vivere un solo anno in piena salute.

16.3.2 L'utilità del denaro

La teoria dell'utilità affonda le sue radici nell'economia, e questa disciplina offre un ovvio candidato per misurare l'utilità: il denaro (o, più specificatamente, il bilancio netto totale di un agente). Il fatto che il denaro si possa scambiare in modo quasi universale per ogni sorta di beni e servizi suggerisce che esso ricopra un ruolo significativo nelle funzioni di utilità degli esseri umani.

Solitamente, a parità di tutte le altre condizioni, un agente tende a preferire più soldi anziché meno. Diciamo che gli agenti mostrano una **preferenza monotona** per il denaro. Questo non significa che il denaro si comporti come una funzione di utilità, perché non dice nulla sulle preferenze riguardanti le *lotterie* che coinvolgono quantità monetarie.

preferenza monotona

Supponiamo che abbiate trionfato sugli avversari in un gioco televisivo. Il presentatore vi offre una scelta: potete far vostro un milione di euro o puntare tutto sul lancio di una mo-

valore monetario atteso

neta. Se uscirà testa perderete tutto, ma se il risultato sarà croce potrete andare a casa con 2,5 milioni. Se siete come la maggior parte della gente, vi rifiuterete di puntare e intascherete il milione. State agendo in maniera irrazionale?

Presumendo che la moneta non sia truccata, il **valore monetario atteso** (EMV, *expected monetary value*) dell'azzardo è $\frac{1}{2}(0) + \frac{1}{2}(2.500.000) = 1.250.000$, più della somma originale di 1.000.000. Ma questo non significa necessariamente che accettare il lancio della moneta sia la decisione migliore. Indichiamo con S_n lo stato in cui si possiede una ricchezza totale di n , e che la vostra ricchezza corrente sia k . Le utilità attese delle due azioni saranno:

$$EU(\text{Accetta}) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.500.000}),$$

$$EU(\text{Rifiuta}) = U(S_{k+1.000.000}).$$

Per determinare cosa fare dobbiamo assegnare dei valori di utilità agli stati corrispondenti agli esiti. L'utilità non è direttamente proporzionale al valore monetario, infatti l'utilità del primo milione è molto alta (o così dicono), mentre l'utilità di ogni milione aggiuntivo è minore. Supponiamo che assegniate un'utilità 5 al vostro stato finanziario corrente (S_k), 9 allo stato $S_{k+2.500.000}$ e 8 allo stato $S_{k+1.000.000}$. In questo caso l'azione razionale sarebbe rifiutare la scommessa, perché la sua utilità attesa è pari solo a 7, un valore inferiore a 8. D'altra parte, un miliardario probabilmente avrà una funzione di utilità localmente lineare su un intervallo di qualche milione, perciò accetterebbe la scommessa.

In un pionieristico studio delle funzioni di utilità reali, Grayson (1960) trovò che l'utilità del denaro era quasi esattamente proporzionale al *logaritmo* della quantità (quest'idea fu suggerita per la prima volta da Bernoulli (1738); cfr. Esercizio 16.STPT). Una particolare curva di utilità, che riguarda un certo Mr. Beard, è riportata nella Figura 16.2(a). I dati ottenuti sulle preferenze di Mr. Beard sono consistenti con una funzione di utilità

$$U(S_{k+n}) = -263,31 + 22,09 \log(n + 150.000)$$

nell'intervallo che va da $n = -150.000$ e $n = 800.000$.

Non dobbiamo pensare che questa sia la funzione di utilità definitiva per i valori monetari, ma è probabile che per la maggior parte delle persone la funzione di utilità sia concava per una ricchezza positiva. Indebitarsi non è bello, ma le preferenze tra diversi livelli di debito possono evidenziare un rovesciamento della concavità associata a una ricchezza positiva. Per esempio, una persona che ha già debiti per 10.000.000 di euro potrebbe accettare di

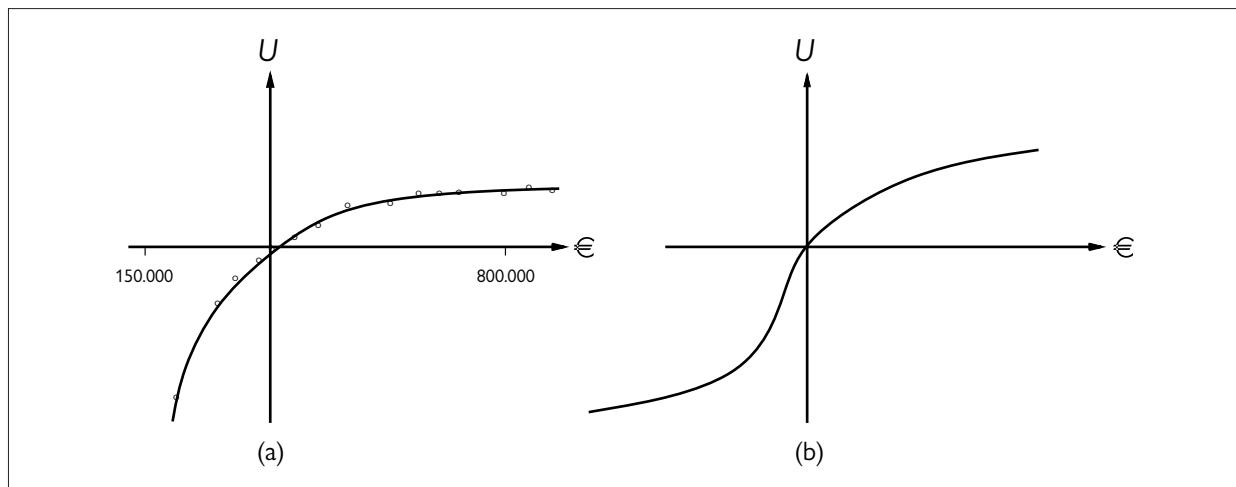


Figura 16.2 L'utilità del denaro. (a) Dati empirici da Mr. Beard su un intervallo limitato. (b) Una tipica curva su tutto il dominio.

scommettere sul lancio di una moneta con la prospettiva di vincere dieci milioni con la testa e perderne venti con la croce.⁴ Questo ci dà la curva a sigmoide della Figura 16.2(b).

Se guardiamo solo la parte positiva delle curve, in cui la pendenza decresce, allora per ogni lotteria L l'utilità di accettare tale lotteria è inferiore a quella di ricevere direttamente con certezza il suo valore monetario atteso:

$$U(L) < U(S_{EMV(L)}).$$

Questo significa che gli agenti che hanno curve di questo tipo sono **avversi al rischio**: preferiscono un guadagno sicuro, anche se è inferiore al valore monetario atteso di una scommessa. D'altro canto, nella regione “dei disperati” con grande ricchezza negativa della Figura 16.2(b), il comportamento è di **propensione al rischio**. Il valore che un agente accetterà al posto di una lotteria è chiamato **equivalente certo** della lotteria. Alcuni studi hanno mostrato che la maggior parte delle persone accetterà una cifra intorno ai 400 euro al posto di una scommessa che ne fornisce 1000 la metà delle volte e nulla nell'altra metà: l'equivalente certo di quella lotteria vale quindi 400 euro, mentre l'EMV è 500 euro.

avverso al rischio

**propensione
al rischio**

equivalente certo

La differenza tra il valore monetario atteso di una lotteria e il suo equivalente certo è chiamata **premio di assicurazione**. L'avversione al rischio è alla base dell'industria delle assicurazioni, perché significa che i premi di assicurazione sono positivi: le persone preferiscono pagare un piccolo premio di assicurazione piuttosto che scommettere il prezzo della loro casa sul fatto che non si verifichi un incendio. Dal punto di vista della compagnia di assicurazioni, il valore della casa è molto piccolo rispetto alle riserve finanziarie totali. Questo significa che la curva di utilità dell'assicuratore è praticamente lineare in quella regione, e la scommessa non costa quasi nulla alla compagnia.

**premio di
assicurazione**

Notate che, per *piccoli* cambiamenti relativamente alla ricchezza corrente posseduta, quasi ogni curva sarà approssimativamente lineare. Si dice che un agente che ha una curva lineare è **neutrale rispetto al rischio**. Nelle scommesse che coinvolgono piccole somme, quindi, ci aspettiamo una neutralità rispetto al rischio. In un certo senso, questo giustifica la procedura semplificata che nel Paragrafo 12.2.3 proponeva di usare scommesse piccole per valutare le probabilità e giustificare i relativi assiomi.

**neutrale rispetto
al rischio**

16.3.3 Utilità attesa e delusione dopo la decisione

La scelta razionale della migliore azione, a^* , corrisponde alla massimizzazione dell'utilità attesa:

$$a^* = \underset{a}{\operatorname{argmax}} EU(a).$$

Se abbiamo calcolato correttamente l'utilità attesa secondo il nostro modello di probabilità, e se tale modello riflette correttamente i processi stocastici di base che generano i risultati, allora, in media, otterremo l'utilità attesa se l'intero processo viene ripetuto molte volte.

Tuttavia, nella realtà il modello solitamente comporta un eccesso di semplificazione della situazione reale, o perché non conosciamo abbastanza la realtà (per esempio quando prendiamo decisioni di investimento complesse) o perché il calcolo della vera utilità attesa è troppo difficile (per esempio quando facciamo una mossa al gioco del backgammon, in cui si dovrebbe tenere conto di tutti i possibili futuri lanci di dadi). In questi casi si lavora in realtà con *stime* $\widehat{EU}(a)$ della vera utilità attesa. Ipotizzeremo, forse con un eccesso di ottimismo, che le stime siano **non distorte**, cioè che il valore atteso dell'errore, $E(\widehat{EU}(a) - EU(a))$, sia zero. In tal caso, appare ragionevole scegliere l'azione con la più alta utilità attesa e attendersi di ricevere tale utilità, in media, con l'esecuzione di tale azione.

stima non distorta

⁴ Un simile comportamento potrebbe essere considerato disperato, ma è razionale se ci si trova già in una situazione disperata.

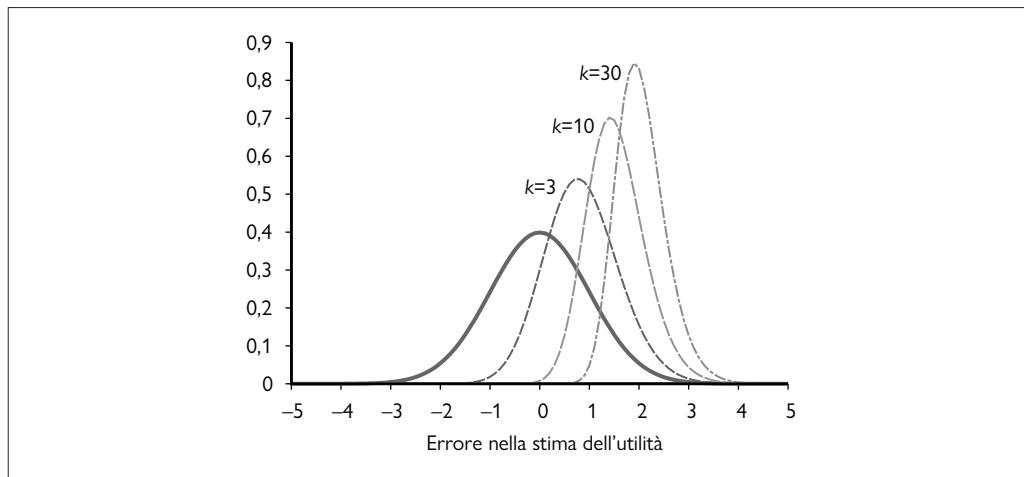


Figura 16.3 Ottimismo ingiustificato causato dalla scelta delle migliori k opzioni: ipotizziamo che ogni opzione abbia una vera utilità di 0, ma una stima di utilità distribuita secondo una normale unitaria (curva più spessa). Le altre curve rappresentano le distribuzioni del massimo di k stime per $k = 3, 10$ e 30 .

Sfortunatamente, l'esito reale sarà di solito significativamente *peggiore* di quello stimato, anche con una stima non distorta. Per capirlo, considerate un problema decisionale in cui vi sono k scelte, ognuna delle quali ha una vera utilità stimata di 0. Supponete che l'errore di ogni stima di utilità sia indipendente e abbia distribuzione normale unitaria, cioè una gaussiana con media zero e deviazione standard 1 (la curva più spessa nella Figura 16.3). Ora, quando cominciamo a generare le stime, otterremo alcuni errori negativi (stime pessimistiche) e alcuni positivi (stime ottimistiche). Poiché selezioniamo l'azione con la stima di utilità *più alta*, favoriamo le stime più ottimistiche, e questo costituisce una fonte di bias o distorsione.

È facile calcolare la distribuzione del massimo delle k stime e quindi quantificare l'entità della nostra delusione (questo calcolo è un caso speciale del calcolo di una **statistica d'ordine**, la distribuzione di un qualsiasi elemento ordinato in un campione). Supponiamo che ogni stima X_i abbia una funzione di densità di probabilità $f(x)$ e una distribuzione cumulata $F(x)$ (come è spiegato nell'Appendice A, la distribuzione cumulata F misura la probabilità che il costo sia minore o uguale a una data somma, cioè è l'integrale della funzione di densità originale f). Ora sia X^* la stima più alta, quindi $\max\{X_1, \dots, X_k\}$. Allora la distribuzione cumulata per X^* è:

$$\begin{aligned} P(\max\{X_1, \dots, X_k\} \leq x) &= P(X_1 \leq x, \dots, X_k \leq x) \\ &= P(X_1 \leq x) \dots P(X_k \leq x) = F(x)^k. \end{aligned}$$

La funzione di densità di probabilità è la derivata della funzione di distribuzione cumulata, perciò la densità per X^* , il valore massimo delle k stime, è:

$$P(x) = \frac{d}{dx}(F(x)^k) = kf(x)(F(x))^{k-1}.$$

Queste densità sono mostrate per diversi valori di k nella Figura 16.3, con riferimento al caso in cui $f(x)$ è la normale unitaria. Per $k = 3$, la densità per X^* ha una media di circa 0,85, perciò la delusione media sarà di circa l'85% della deviazione standard delle stime di utilità. Con un maggior numero di scelte, è più probabile che si presentino stime estremamente ottimistiche: per $k = 30$, la delusione sarà circa il doppio della deviazione standard delle stime.

Il fatto che la stima di utilità attesa per la migliore scelta tenda a essere troppo elevata è detto **maledizione dell'ottimizzatore** (Smith e Winkler, 2006) e colpisce anche i più esperti analisti decisionali e statistici. Tra le manifestazioni più gravi vi sono il credere che un nuovo

farmaco che ha curato l'80% dei pazienti in una sperimentazione curi l'80% dei pazienti anche nella realtà (quel farmaco è stato scelto tra k = migliaia di altri farmaci candidati), o credere che un fondo comune di investimento pubblicizzato per aver ottenuto rendimenti superiori alla media continuerà ad avere rendimenti simili anche in futuro (è stato scelto appositamente per la pubblicità tra k = decine di altri fondi presenti nel portafoglio generale dell'azienda). Può anche darsi che ciò che appare come migliore scelta in realtà non lo sia, se la varianza della stima di utilità è alta: un farmaco che ha curato 9 pazienti su 10 ed è stato scelto tra migliaia sperimentati è probabilmente *peggiore* di un altro che ne curati 800 su 1000.

La maledizione dell'ottimizzatore si presenta spesso perché i processi di selezione basati sulla massimizzazione dell'utilità attesa sono applicati ovunque, perciò considerare le stime di utilità “alla lettera” è una cattiva idea. Possiamo evitare questa minaccia adottando un approccio bayesiano che utilizzi un modello di probabilità esplicito $\mathbf{P}(\overline{EU} | EU)$ dell'errore delle stime di utilità. Dati questo modello e una distribuzione a priori delle utilità che possiamo ragionevolmente considerare attese, prendiamo la stima dell'utilità come evidenza e calcoliamo la distribuzione a posteriori della vera utilità utilizzando la regola di Bayes.

16.3.4 Giudizio umano e irrazionalità

La teoria delle decisioni è una **teoria normativa**: descrive come un agente razionale *dovrebbe* agire. Una **teoria descrittiva**, invece, descrive come gli agenti effettivi – per esempio, umani – agiscono effettivamente. L'applicazione alla teoria economica risulterebbe molto facilitata se teoria normativa e descrittiva coincidessero, ma esistono prove sperimentali del contrario. Le evidenze suggeriscono che gli esseri umani siano “prevedibilmente irrazionali” (Ariely, 2009).

Il problema più noto è il paradosso di Allais (Allais, 1953): alle persone viene data la possibilità di scegliere tra le lotterie A e B e poi tra C e D :

- | | |
|-----------------------------------------|------------------------------------------|
| A: 80% di possibilità di vincere €4000 | C: 20% di possibilità di vincere €4000 |
| B: 100% di possibilità di vincere €3000 | D: 25% di possibilità di vincere €3000 . |

La maggior parte delle persone preferisce B ad A (per andare sul sicuro) e C a D (per considerare l'EMV più alto). L'analisi normativa non è d'accordo. Per capirlo, utilizziamo la libertà implicata dall'Equazione (16.2) di assegnare $U(€0) = 0$; in questo caso, $B > A$ implica che $U(€3000) > 0,8U(€4000)$, mentre $C > D$ implica esattamente il contrario. In altre parole, non sembra esserci alcuna funzione di utilità consistente con queste scelte.

Una possibile spiegazione per le preferenze apparentemente irrazionali è data dall'**effetto certezza** (Kahneman e Tversky, 1979): le persone sono fortemente attratte da guadagni certi, per diversi motivi.

In primo luogo, le persone preferiscono ridurre il loro carico di lavoro mentale: scegliendo risultati certi, non devono preoccuparsi di calcolare probabilità. Ma l'effetto persiste anche quando i calcoli sono molto semplici.

In secondo luogo, le persone a volte non si fidano delle probabilità dichiarate. Mi fido che il lancio di una moneta sia circa 50/50 se ho il controllo sulla moneta e sul lancio, ma tendo a non fidarmi se il lancio è effettuato da qualcuno che ha un interesse nel risultato.⁵ Se non c'è fiducia, è meglio andare sul sicuro.⁶

teoria normativa
teoria descrittiva

effetto certezza

⁵ Per esempio, il matematico e prestigiatore Persi Diaconis è in grado di lanciare una moneta e far comparire la faccia desiderata quando vuole (Landhuis, 2004).

⁶ Anche andando sul sicuro non è detto che vi sia certezza. Nonostante tutte le promesse, non abbiamo ancora ricevuto quei 27 milioni di euro dal conto su una banca nigeriana intestato a un parente deceduto di cui non sapevamo nulla prima.

In terzo luogo, le persone possono considerare il loro stato emotivo oltre a quello finanziario. Sanno che subirebbero un dispiacere se rinunciassero a un premio certo (*B*) per una possibilità dell’80% di un premio più alto che però venisse perso.

In altre parole, se viene scelto *A*, allora c’è una probabilità del 20% di non ricevere denaro e sentirsi un completo idiota, cosa peggiore di non ricevere denaro e basta. Quindi, forse le persone che scelgono *B* preferendolo ad *A* e *C* preferendolo a *D* non sono irrazionali: sono disposte a rinunciare a 200 euro di valore monetario atteso per evitare una probabilità del 20% di sentirsi idioti.

Un problema correlato è il paradosso di Ellsberg. In questo caso i premi sono fissati, ma ci sono meno vincoli sulle probabilità. Il guadagno dipenderà dal colore di una pallina scelta da un’urna. Sapete che l’urna contiene 1/3 di palline rosse e 2/3 di palline nere o gialle, ma non sapete quante nere e quante gialle. Anche qui vi si chiede se preferiate la lotteria *A* o *B*; e poi *C* o *D*:

A : €100 per una pallina rossa
B : €100 per una pallina nera

C : €100 per una pallina rossa o gialla
D : €100 per una pallina nera o gialla.

Dovrebbe risultare chiaro che, se pensate che vi siano più palline rosse che nere, dovreste preferire *A* rispetto a *B* e *C* rispetto a *D*; se invece pensate che vi siano meno palline rosse che nere dovreste preferire l’opposto. Tuttavia, la maggior parte delle persone preferisce *A* rispetto a *B* e *D* rispetto a *C*, anche se non esiste uno stato del mondo per cui tali preferenze sono razionali. Pare che le persone abbiano un’**avversione all’ambiguità**: *A* fornisce una probabilità di 1/3 di vincere, mentre *B* può offrire una probabilità qualsiasi tra 0 e 2/3. Similmente, *D* fornisce una probabilità di 2/3, mentre *C* un valore qualsiasi tra 1/3 e 3/3. La maggior parte delle persone sceglie la probabilità nota preferendola all’ignota.

avversione all’ambiguità

effetto framing

Un altro problema è che i termini esatti in cui è espresso un problema decisionale possono avere un notevole impatto sulle scelte dell’agente; questo è il cosiddetto **effetto framing**. Studi sperimentali mostrano che una procedura medica descritta con una frase come “ha un tasso di sopravvivenza del 90%” viene apprezzata circa il doppio di una descritta con “ha un tasso di mortalità del 10%”, anche se in realtà le due frasi significano esattamente la stessa cosa. Questa discrepanza di giudizio è stata rilevata in numerosi esperimenti e rimane più o meno invariata con soggetti che sono pazienti ricoverati in ospedale, studenti di materie statistiche o medici esperti.

effetto ancoraggio

Le persone si sentono più a loro agio con giudizi di utilità *relativi* piuttosto che assoluti. In genere non so bene quanto potrebbero piacermi i vari vini offerti da un ristorante, e il ristoratore sfrutta questo fatto offrendomi una bottiglia da 200 euro che nessuno acquisterà, ma che serve per distorcere verso l’alto il valore stimato dal cliente di tutti i vini del ristorante, facendo sembrare un affare una bottiglia da 55 euro. Questo è il cosiddetto **effetto ancoraggio**.

Se gli informatori umani insistono nel fornire giudizi contraddittori sulle preferenze, non c’è nulla che gli agenti automatizzati possano fare per essere consistenti con esse. Fortunatamente, spesso gli esseri umani sono pronti a rivedere i propri giudizi di preferenza alla luce di nuove considerazioni. Paradossi come quelli di Allais e di Ellsberg si riducono notevolmente (senza però scomparire) se le scelte sono spiegate meglio. In un lavoro svolto alla Harvard Business School per valutare l’utilità del denaro, Keeney e Raiffa (1976, p. 210) determinarono quanto segue:

I soggetti tendono a essere troppo avversi al rischio nei casi di piccola scala e quindi ... le funzioni di utilità ricostruite esibiscono premi eccessivamente grandi per le lotterie con una gamma ampia di risultati. ... La maggior parte dei soggetti, comunque, può superare le proprie inconsistenze e sentire di aver imparato un’importante lezione riguardo al modo in cui desiderano comportarsi. Di conseguenza, alcuni soggetti disdicono la loro assicurazione sui danni automobilistici e sottoscrivono assicurazioni sulla vita.

Le prove dell’irrazionalità degli esseri umani sono messe in discussione anche dai ricercatori nel campo della **psicologia evoluzionista**, che sottolineano il fatto che i meccanismi decisionali del nostro cervello non si sono evoluti per risolvere problemi con probabilità e premi descritti da numeri decimali. Diamo per buona l’ipotesi che il cervello disponga di meccanismi neurali integrati per eseguire calcoli con probabilità e utilità, o qualcosa di funzionalmente equivalente. Allora, gli input richiesti sarebbero ottenuti attraverso il cumulo di esperienze relative a risultati e premi, anziché attraverso valori numerici presentati attraverso il linguaggio naturale.

**psicologia
evoluzionista**

Non è affatto scontato che possiamo accedere direttamente ai meccanismi neurali del nostro cervello presentando problemi decisionali in forma linguistica/numerica. Il fatto stesso che formulazioni diverse dello *stesso problema decisionale* portino a scelte diverse suggerisce che il problema decisionale in realtà non viene comunicato adeguatamente. Stimolati da questa osservazione, gli psicologi hanno cercato di presentare i problemi decisionali e di ragionamento incerto in forme “evoluzionisticamente appropriate”; per esempio, anziché dire: “tasso di sopravvivenza del 90%”, lo sperimentatore potrebbe mostrare 100 animazioni stilizzate dell’intervento medico, con il paziente che muore in 10 di esse e sopravvive in 90. Quando i problemi decisionali sono presentati in questo modo, il comportamento delle persone sembra molto più vicino allo standard della razionalità.

16.4 Funzioni di utilità multiattributo

Nel campo delle politiche pubbliche, le decisioni possono comportare spese elevate in termini di denaro e anche di vite umane. Per esempio, nel decidere quali livelli di emissioni pericolose consentire a un impianto per la produzione di energia, i politici devono valutare da un lato la prevenzione di morti e disabilità, dall’altro il beneficio dell’energia e il costo economico di mitigare le emissioni. Per scegliere il terreno dove costruire un nuovo aeroporto si devono considerare i disagi causati dal cantiere, il costo del terreno, la distanza dai centri popolati, il rumore causato dagli aerei, i problemi di sicurezza legati alla topografia locale e al clima e così via. Problemi come questi, in cui gli esiti sono caratterizzati da due o più attributi, sono l’argomento della **teoria dell’utilità multiattributo**, che è in sostanza la teoria di confrontare mele e arance.

**teoria dell’utilità
multiattributo**

Siano $\mathbf{X} = X_1, \dots, X_n$ gli attributi e $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ un vettore completo di assegnamenti, dove ogni x_i è un valore numerico o un valore discreto con un ordinamento assunto sui valori. L’analisi è più facile se li disponiamo in modo che a valori più alti dell’attributo corrisponda sempre un’utilità maggiore: le utilità sono monotonicamente crescenti. Questo significa che non possiamo usare, per esempio, il numero di morti d come attributo: dovremmo usare $-d$. Significa anche che non possiamo usare come attributo la temperatura in una stanza, t . Se la funzione di utilità per la temperatura ha un picco a 21°C e cala monotonicamente su entrambi i lati di tale picco, potremmo suddividere l’attributo in due parti, usando $t - 21$ per misurare se la stanza è abbastanza calda, e $21 - t$ per misurare se è abbastanza fresca. Entrambi questi attributi sarebbero monotonicamente crescenti fino a raggiungere il loro valore di utilità massima in 0; da quel punto in poi la curva di utilità è piatta, a indicare che non si ottiene più “abbastanza caldo” sopra i 21°C , né “abbastanza fresco” sotto i 21°C .

Gli attributi per il problema dell’aeroporto potrebbero essere i seguenti:

- *Traffico*, misurato con il numero di voli al giorno;
- *Sicurezza*, misurato con il negativo del numero atteso di morti per anno;
- *Quietè*, misurato con il negativo del numero di persone che vivono sotto le rotte degli aerei;
- *Frugalità*, misurato con il costo negativo di costruzione.

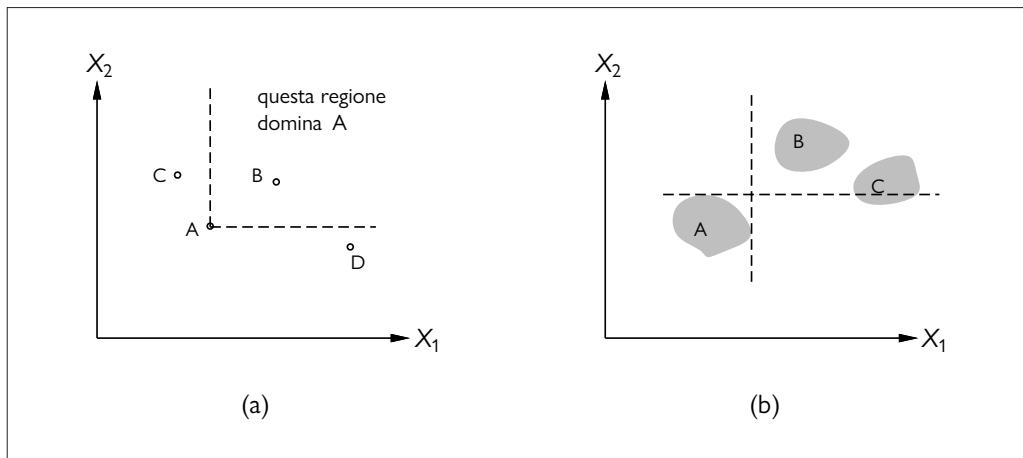


Figura 16.4 Dominanza stretta. (a) Deterministica: l'opzione A è strettamente dominata da B, ma non da C o D. (b) Incerta: A è dominata strettamente da B ma non da C.

Cominceremo a esaminare i casi in cui è possibile prendere una decisione *senza* combinare i valori degli attributi in un singolo valore di utilità; quindi passeremo ai casi in cui è possibile esprimere in modo molto conciso l'utilità di una combinazione di attributi.

16.4.1 Dominanza

dominanza stretta

Supponiamo che la posizione candidata S_1 per il nostro aeroporto costi meno, generi meno inquinamento acustico e sia più sicura della posizione S_2 : in questo caso non avremmo dubbi a rifiutare S_2 . Possiamo dire che c'è una **dominanza stretta** di S_1 su S_2 . In generale, se tutti gli attributi di un'opzione hanno un valore inferiore a quelli di un'altra, non è necessario considerare ulteriormente la prima opzione. La dominanza stretta è spesso utile per sfoltire il campo delle scelte riducendolo alle possibilità veramente interessanti, benché raramente sia in grado di fornire alla fine una soluzione unica. La Figura 16.4(a) mostra un diagramma schematico in un caso con due soli attributi.

Tutto questo va bene nel caso deterministico, in cui i valori degli attributi sono noti con certezza: ma che dire del caso generale, in cui gli esiti sono incerti? Ebbene, nonostante l'incertezza è ancora possibile costruire l'equivalente della dominanza stretta, in cui tutti i possibili esiti concreti di S_1 dominano strettamente tutti i possibili esiti di S_2 (Figura 16.4(b)). Naturalmente, questo accadrà ancor più raramente che nel caso deterministico.

dominanza stocastica

Fortunatamente esiste una generalizzazione più utile, chiamata **dominanza stocastica**, che nei problemi reali si verifica abbastanza frequentemente. Per comprenderla facilmente facciamo riferimento al caso con un solo attributo. Supponiamo di credere che il costo di costruire l'aeroporto nella posizione S_1 sia uniformemente distribuito tra 2,8 e 4,8 miliardi di euro, mentre per la posizione S_2 il costo è sempre distribuito uniformemente, ma fra 3 e 5,2 miliardi. Definiamo l'attributo *Frugalità* come il negativo di tale costo. La Figura 16.5(a) riporta le distribuzioni per la frugalità dei siti S_1 e S_2 . Allora, partendo solo dall'informazione che la scelta più frugale è migliore (a parità di tutte le altre condizioni), possiamo dire che S_1 domina stocasticamente S_2 (e quindi il sito S_2 può essere scartato). È importante notare che questa *non* è una conseguenza del confronto tra i costi attesi: se avessimo saputo che il costo di S_1 era, poniamo, *esattamente* 3,8 miliardi, saremmo stati *impossibilitati* a scegliere senza ricevere informazioni aggiuntive sull'utilità del denaro. Potrebbe sembrare strano che disporre di *più* informazioni sul costo di S_1 possa rendere l'agente *meno* capace di decidere.

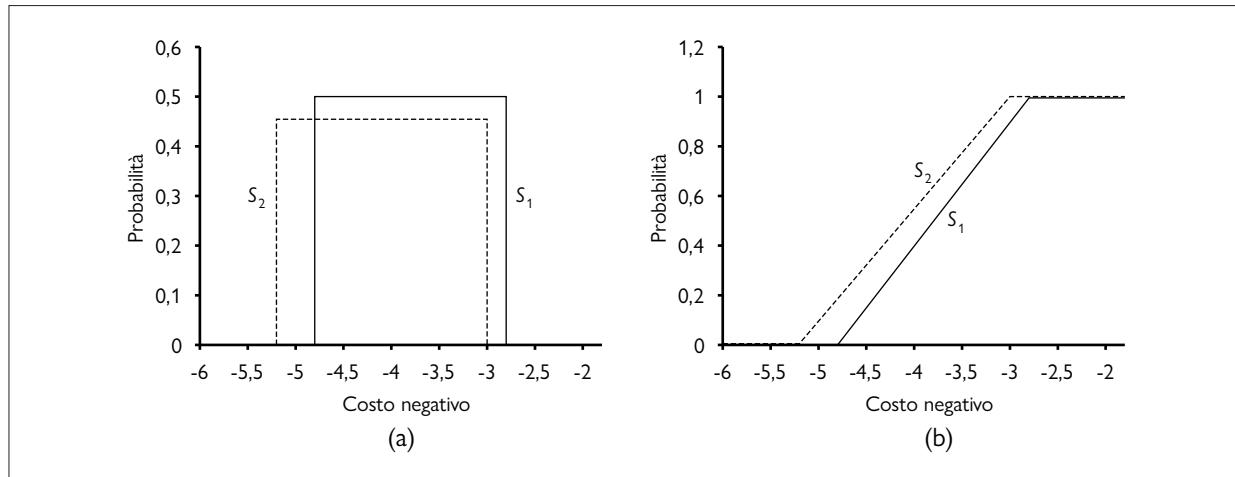


Figura 16.5 Dominanza stocastica. (a) S_1 domina stocasticamente S_2 sulla frugalità (costo negativo). (b) Distribuzioni cumulate della frugalità di S_1 e S_2 .

Il paradosso è risolto notando che, in assenza di informazioni esatte sui costi, la decisione è più facile da prendere ma ha una probabilità maggiore di essere sbagliata.

La relazione esatta tra le distribuzioni degli attributi affinché ci sia una dominanza stocastica si comprende meglio considerando le distribuzioni cumulate, riportate nella Figura 16.5(b). Se la distribuzione cumulata di S_1 sta sempre alla destra di quella di S_2 , si può dire che stocasticamente S_1 è una soluzione più economica di S_2 . Formalmente, se due azioni A_1 e A_2 hanno distribuzioni di probabilità $p_1(x)$ e $p_2(x)$ sull'attributo X , allora A_1 domina stocasticamente A_2 su X se:

$$\forall x \int_{-\infty}^{\infty} p_1(x') dx' \leq \int_{-\infty}^x p_2(x') dx' .$$

L'importanza di questa definizione per prendere decisioni ottime deriva dalla seguente proprietà: *se A_1 domina stocasticamente A_2 , allora l'utilità attesa di A_1 è almeno pari a quella di A_2 per qualsiasi funzione di utilità $U(x)$ monotonicamente non decrescente*. Per capirlo, consideriamo le due utilità attese, $\int p_1(x)U(x)dx$ e $\int p_2(x)U(x)dx$. Inizialmente non è ovvio il motivo per cui il primo integrale è maggiore del secondo, dato che la condizione di dominanza stocastica ha un integrale di p_1 minore dell'integrale di p_2 .



Tuttavia, anziché pensare all'integrale su x , pensiamo all'integrale su y , la probabilità cumulata, come illustrato nella Figura 16.5(b). Per qualsiasi valore di y , il valore corrispondente di x (e quindi di $U(x)$) è maggiore per S_1 che per S_2 ; quindi, se integriamo una quantità più grande sull'intero intervallo di y , otteniamo per forza un risultato più grande. Formalmente si tratta solo di sostituire $y = P_1(x)$ nell'integrale per il valore atteso di S_1 e $y = P_2(x)$ nell'integrale per il valore atteso di S_2 . Con queste sostituzioni abbiamo $dy = \frac{d}{dx}(P_1(x))dx = p_1(x)dx$ per S_1 e $dy = p_2(x)dx$ per S_2 , quindi

$$\int_{-\infty}^{\infty} p_1(x) U(x) dx = \int_0^1 U(P_1^{-1}(y)) dy \geq \int_0^1 U(P_2^{-1}(y)) dy = \int_{-\infty}^{\infty} p_2(x) U(x) dx .$$

Questa diseguaglianza ci consente di preferire A_1 ad A_2 in un problema monoattributo. Più in generale, se un'azione è stocasticamente dominata da un'altra su *tutti* gli attributi in un problema multiattributo, allora può essere scartata.

La condizione di dominanza stocastica potrebbe sembrare una nozione alquanto tecnica, difficile da valutare senza complessi calcoli probabilistici. In realtà, in molti casi può essere determinata molto facilmente. Per esempio, preferireste cadere a testa in giù su un pavimento di cemento da un'altezza di 3 millimetri o di 3 metri? Pensiamo che abbiate scelto 3 millimetri – buona scelta! Ma perché questa è necessariamente una decisione migliore? C'è un buon grado di incertezza su quale livello di danno rischiate di farvi in entrambi i casi, ma per ogni livello di danno dato, la probabilità che raggiungerete almeno tale livello è più alta se cadete da 3 metri rispetto a 3 millimetri. In altri termini, la caduta da 3 millimetri domina stocasticamente la caduta da 3 metri sull'attributo *Sicurezza*.

Questo tipo di ragionamento è naturale per gli esseri umani; è talmente ovvio che non ci pensiamo nemmeno. La dominanza stocastica è presente anche nel problema dell'aeroporto. Supponiamo per esempio che il costo dei trasporti per la costruzione dipenda dalla distanza dalla sede del fornitore dei materiali. Il costo di per sé è incerto, ma all'aumentare della distanza, aumenta anch'esso. Se il sito S_1 è più vicino di S_2 , allora S_1 dominerà S_2 sulla frugalità. Esistono algoritmi (che non presenteremo qui) per propagare questo tipo di informazione qualitativa tra variabili incerte per mezzo di **reti probabilistiche qualitative**, che permettono a un sistema di prendere decisioni razionali in base alla dominanza stocastica senza utilizzare alcun valore numerico.

**rete probabilistica
qualitativa**

16.4.2 Struttura delle preferenze e utilità multiattributo

Supponiamo di avere n attributi, ognuno con d possibili valori distinti. Per specificare la funzione di utilità completa $U(x_1, \dots, x_n)$, nel caso pessimo, sono necessari d^n valori. La teoria dell'utilità multiattributo mira a individuare la presenza di un livello strutturale in più nelle preferenze umane, grazie al quale non sia necessario specificare tutti i singoli d^n valori. Avendo identificato una regolarità nelle preferenze, possiamo derivare **teoremi di rappresentazione** per mostrare che un agente con un certo tipo di struttura delle preferenze ha una funzione di utilità

$$U(x_1, \dots, x_n) = F[f_1(x_1), \dots, f_n(x_n)],$$

dove F è (nella nostra speranza) una funzione semplice come l'addizione. Notate l'analogia con l'uso delle reti bayesiane per scomporre la probabilità congiunta di molte variabili casuali.

Come esempio, supponiamo che ogni x_i sia l'importo di denaro a disposizione dell'agente in una particolare valuta: dollari, euro, sterline, ecc. Allora le funzioni f_i potrebbero convertire ciascun importo in una valuta comune, e F sarebbe la semplice addizione di tali importi.

**indipendenza
delle preferenze**

Preferenze senza incertezza

Cominciamo con il caso deterministico. Nel Paragrafo 16.2.2 si è osservato che negli ambienti deterministici l'agente ha una funzione valore che scriviamo qui come $V(x_1, \dots, x_n)$; l'obiettivo è rappresentare tale funzione in modo conciso. La regolarità principale che si riscontra nelle strutture di preferenze deterministiche è chiamata **indipendenza delle preferenze**. Due attributi X_1 e X_2 sono indipendenti per la preferenza da un terzo attributo X_3 se la preferenza tra i due risultati $\langle x_1, x_2, x_3 \rangle$ e $\langle x'_1, x'_2, x_3 \rangle$ non dipende dal particolare valore x_3 dell'attributo X_3 .

Tornando al nostro esempio dell'aeroporto, in cui dobbiamo considerare (tra gli altri attributi) *Quiete*, *Frugalità* e *Sicurezza*, si potrebbe proporre che *Quiete* e *Frugalità* siano preferenzialmente indipendenti da *Sicurezza*. Per esempio, se preferiamo un esito che prevede 20.000 persone residenti lungo le rotte di volo e un costo di costruzione di 4 miliardi a uno che prevede 70.000 persone lungo le rotte di volo e costa 3,7 miliardi, quando il livello di sicurezza in entrambi i casi è di 0,006 morti per miliardo di miglia percorse dai passeggeri, allora arriveremmo alla stessa conclusione anche se il livello di sicurezza fosse 0,012 o 0,003;

e la stessa indipendenza sarebbe verificata per ogni altra coppia di valori di *Quiete* e *Frugalità*. Appare anche evidente che *Frugalità* e *Sicurezza* sono preferenzialmente indipendenti da *Quiete* e che *Quiete* e *Sicurezza* sono parimenti indipendenti da *Frugalità*.

In questo caso diciamo che l'insieme di attributi $\{\text{Quiete}, \text{Frugalità}, \text{Sicurezza}\}$ esibisce una **indipendenza mutua per la preferenza** (MPI, *mutual preferential independence*). L'MPI afferma che, indipendentemente dall'importanza che può avere ogni attributo, esso non influenza in alcun modo i rapporti da gli altri attributi dell'insieme.

L'indipendenza mutua per la preferenza può sembrare un'espressione complicata, ma conduce a una forma semplice per la funzione valore dell'agente (Debreu, 1960): *se gli attributi X_1, \dots, X_n sono mutuamente indipendenti per la preferenza, le preferenze dell'agente possono essere rappresentate da una funzione valore:*

$$V(x_1, \dots, x_n) = \sum_i V_i(x_i).$$

in cui ogni V_i si riferisce solo all'attributo X_i . Per esempio, potrebbe darsi che si possa arrivare a una decisione riguardo alla posizione dell'aeroporto utilizzando la funzione valore

$$V(\text{quiete}, \text{frugalità}, \text{sicurezza}) = -\text{quiete} \times 10^4 - \text{frugalità} - \text{sicurezza} \times 10^{12}.$$

Una funzione valore di questo tipo è chiamata **funzione valore additiva**. Le funzioni additive sono un modo estremamente naturale di descrivere le preferenze di un agente e sono applicabili in molte situazioni reali. Nel caso in cui vi sono n attributi, per determinare una funzione valore additiva è necessario determinare n funzioni valore monodimensionali separate, anziché una sola funzione n -dimensionale; generalmente questo rappresenta una riduzione esponenziale del numero di esperimenti di preferenza necessari. Anche nei casi in cui non vale strettamente la MPI, come in corrispondenza dei valori estremi degli attributi, una funzione valore additiva potrebbe comunque rappresentare una buona approssimazione delle preferenze dell'agente. Ciò è particolarmente vero quando le violazioni alla MPI avvengono in corrispondenza a valori degli attributi che si incontrano raramente nella pratica.

Per capire meglio l'MPI, può essere utile esaminare i casi in cui *non* vale. Supponete di trovarvi in un mercato medievale, intenti a riflettere sull'acquisto di cani da caccia, polli e gabbie di vimini per questi ultimi. I cani da caccia valgono molto, ma se non avete abbastanza gabbie per i polli, questi saranno mangiati dai cani; quindi, il bilanciamento tra cani e polli dipende fortemente dal numero di gabbie, e l'MPI è violata. L'esistenza di queste interazioni tra i vari attributi rende molto più difficile determinare la funzione valore complessiva.

Preferenze in condizioni di incertezza

Quando nel dominio è presente dell'incertezza, è necessario considerare anche la struttura delle preferenze tra le lotterie e comprendere le proprietà risultanti delle funzioni di utilità, anziché delle sole funzioni valore. Dal punto di vista matematico il problema può diventare alquanto complicato, ragion per cui ci limitiamo a presentare uno dei risultati principali per dare almeno un'idea di quel che è possibile fare.

La nozione base è l'**indipendenza tra utilità**, che estende alle lotterie l'indipendenza delle preferenze: un insieme di attributi **X** è indipendente per l'utilità da un insieme di attributi **Y** se le preferenze tra le lotterie che coinvolgono gli attributi in **X** sono indipendenti dai particolari valori di quelli in **Y**. Un insieme di attributi è **mutuamente indipendente per l'utilità** (MUI, *mutually utility independent*) se ognuno dei suoi sottoinsiemi è indipendente per l'utilità da tutti gli attributi rimanenti. Ancora una volta, sembra ragionevole affermare che gli attributi del problema dell'aeroporto siano MUI.

La mutua indipendenza per l'utilità implica che il comportamento dell'agente può essere descritto mediante una **funzione di utilità moltiplicativa** (Keeney, 1974). La forma generale

indipendenza
mutua per la
preferenza



funzione valore
additiva

indipendenza
tra utilità

mutuamente
indipendente
per l'utilità

funzione di utilità
moltiplicativa

di una funzione di utilità moltiplicativa si può apprezzare più facilmente guardando il caso a tre attributi. Scriviamo per brevità U_i per indicare $U_i(x_i)$:

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3.$$

Sebbene non appaia particolarmente semplice, la formula contiene solo tre funzioni di utilità monoattributo e tre costanti. In generale, un problema a n attributi con la proprietà MUI può essere modellato usando n utilità monoattributo e n costanti. Ogni funzione di utilità può essere sviluppata indipendentemente dagli altri attributi, ed è garantito che questa combinazione darà origine alle preferenze globali corrette. Se sono verificate ulteriori ipotesi, si potrà ottenere una funzione di utilità puramente additiva.

16.5 Reti di decisione

**diagramma
di influenza
rete di decisione**

In questo paragrafo presentiamo un meccanismo generale per prendere decisioni razionali. La notazione è spesso chiamata **diagramma di influenza** (Howard e Matheson, 1984), ma noi useremo il termine più descrittivo **rete di decisione**. Le reti di decisione arricchiscono le reti bayesiane con tipi di nodo aggiuntivi per le azioni e le utilità. Come esempio continuiamo a usare il problema di scegliere un sito dove costruire un aeroporto.

16.5.1 Rappresentare un problema con una rete di decisione

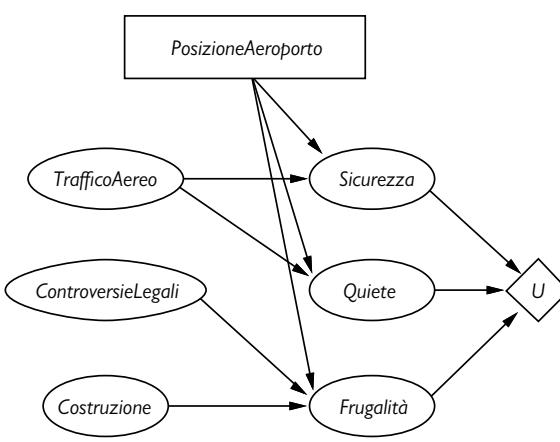
nodo di causalità

Nella sua forma più generale, una rete di decisione rappresenta informazioni sullo stato corrente dell'agente, le sue possibili azioni, gli stati risultanti da esse e la loro utilità; di conseguenza fornisce una base ideale per implementare agenti basati sull'utilità del tipo che abbiamo descritto nel Paragrafo 2.4.5. La Figura 16.6 mostra una rete di decisione per il problema dell'aeroporto, che esemplifica i tre tipi di nodi utilizzati.

- **I nodi di causalità** (ovali) rappresentano variabili casuali, proprio come nelle reti bayesiane. L'agente potrebbe essere incerto riguardo al costo di costruzione, il livello di traffico aereo e delle conseguenti controversie legali, e le variabili *Sicurezza*, *Quiete* e *Frugalità*, ognuna delle quali dipende anche dalla posizione scelta. A ogni nodo di causalità è associata una distribuzione condizionata indicizzata in base allo stato dei nodi genitori. Oltre ai nodi di causalità, i genitori possono includere anche nodi di decisione. Notate che ogni nodo di causalità dello stato corrente potrebbe far parte di una grande rete ba-

Figura 16.6

Una rete di decisione per il problema del sito in cui costruire l'aeroporto.



yesiana per la valutazione dei costi di costruzione, del livello di traffico aereo o delle possibilità di controversie legali.

- **I nodi di decisione** (rettangoli) rappresentano punti in cui si deve scegliere un'azione. In questo caso l'azione *PosizioneAeroporto* può assumere un valore diverso per ogni sito preso in considerazione. La scelta influenza la frugalità, la sicurezza e la quiete della soluzione. In questo capitolo ci occupiamo di reti con un singolo nodo di decisione; nel Capitolo 17 tratteremo i casi in cui è necessario prendere più di una decisione.
- **I nodi di utilità** (rombi) rappresentano la funzione di utilità dell'agente.⁷ Un nodo di utilità ha come genitori tutte le variabili che descrivono gli esiti che influenzano direttamente l'utilità. A esso è associata una descrizione dell'utilità dell'agente in funzione degli attributi dei genitori. La descrizione potrebbe consistere semplicemente in una tabulazione della funzione, ma potrebbe anche essere una funzione parametrizzata additiva o lineare dei valori degli attributi. Per ora ipotizziamo che la funzione sia deterministica, cioè che, dati i valori delle variabili genitori, il valore del nodo di utilità sia completamente determinato.

In molti casi si usa anche una forma semplificata. La notazione rimane identica, ma i nodi di causalità che descrivono gli esiti sono omessi; il nodo di utilità è collegato direttamente al nodo di decisione e ai nodi di stato corrente. In questo caso, invece di rappresentare una funzione di utilità sugli stati corrispondenti agli esiti, il nodo di utilità rappresenta l'utilità *attesa* associata a ogni azione, definita nell'Equazione (16.1), cioè al nodo è associata una **funzione azione-utilità** (detta anche **Q-function** o **funzione-Q**) nell'apprendimento con rinfoco, come vedremo nel Capitolo 22 del Volume 2). La Figura 16.7 mostra la rappresentazione azione-utilità del problema dell'aeroporto.

Notate che, dal momento che i nodi di causalità *Quietè*, *Sicurezza* e *Frugalità* nella Figura 16.6 si riferiscono a stati futuri, i loro valori non potranno mai servire come variabili di evidenza. Quindi, la versione semplificata che li omette può essere usata ogni volta che può essere utilizzata la forma più generale. Benché la forma semplificata contenga meno nodi, l'omissione di una descrizione esplicita dell'esito della decisione la rende meno flessibile rispetto ai cambiamenti. Per esempio, nella Figura 16.6 un cambiamento nei livelli di rumosità degli aerei può essere espresso modificando la tabella delle probabilità condizionate

nodo di decisione

nodo di utilità

**funzione
azione-utilità**

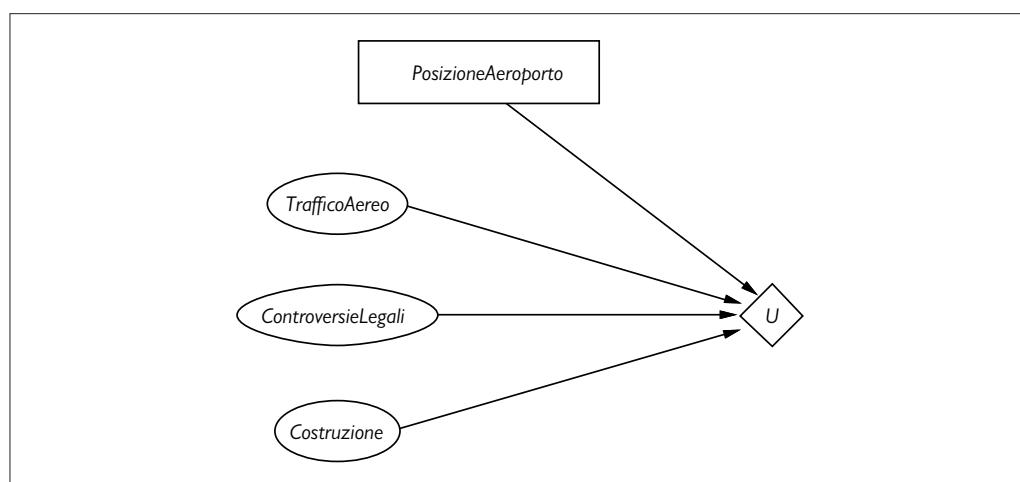


Figura 16.7

Una rappresentazione semplificata del problema del sito in cui costruire l'aeroporto. I nodi di causalità degli stati corrispondenti agli esiti sono stati eliminati mediante fattorizzazione.

⁷ Nella letteratura, questi nodi sono spesso chiamati **nodi valore**.

del nodo *Quiete*, e un cambiamento del peso associato all'inquinamento acustico nella funzione di utilità può essere espresso modificando la tabella delle utilità. Nel diagramma azione-utilità della Figura 16.7, d'altro canto, tutti questi cambiamenti dovranno obbligatoriamente riflettersi in altrettante modifiche della tabella azione-utilità. Essenzialmente, si può dire che la formulazione azione-utilità sia una versione *compilata* di quella originale, ottenuta mediante *summing out* delle variabili dello stato risultato.

16.5.2 Valutazione delle reti di decisione

Le azioni sono selezionate valutando la rete per ogni possibile configurazione del nodo di decisione. Una volta che il suo valore è fissato, si comporta esattamente come un nodo di causalità che è stato scelto come variabile di evidenza. L'algoritmo di valutazione è il seguente.

1. Imposta le variabili di evidenza per lo stato corrente.
2. Per ogni possibile valore del nodo decisione:
 - a. assegna tale valore al nodo decisione;
 - b. calcola le probabilità a posteriori dei nodi genitori del nodo di utilità con un algoritmo standard di inferenza probabilistica;
 - c. calcola l'utilità risultante dell'azione.
3. Restituisci l'azione con l'utilità più alta.

Questo è un approccio diretto che può utilizzare qualsiasi algoritmo per reti bayesiane disponibile, e può essere incorporato direttamente nel progetto di agente della Figura 12.1. Come vedremo nel Capitolo 17, la possibilità di eseguire una sequenza di più azioni rende il problema molto più interessante.

16.6 Il valore dell'informazione



teoria del valore dell'informazione

Nella precedente analisi abbiamo dato per scontato che tutta l'informazione rilevante, o almeno quella disponibile, fosse fornita all'agente prima della sua decisione. Nella pratica, questo non avviene quasi mai. *Una delle parti più importanti del processo decisionale è sapere quali domande porre.* Per esempio, un dottore non può aspettarsi di ricevere i risultati di *tutti i possibili* esami diagnostici nel momento in cui un paziente gli si presenta per la prima volta. Gli esami sono spesso costosi e talvolta pericolosi (sia direttamente che per i ritardi da essi causati). La loro importanza dipende da due fattori: la prima cosa da considerare è se il risultato di un esame porterebbe a una cura significativamente migliore, il secondo è la probabilità di correttezza dei risultati stessi.

Questo paragrafo descrive la **teoria del valore dell'informazione**, che permette a un agente di scegliere quali informazioni ottenere. Ipotizziamo che prima di selezionare un'azione "reale" rappresentata dal nodo di decisione, l'agente possa acquisire il valore di ognuna delle variabili casuali potenzialmente osservabili nel modello. Quindi la teoria del valore dell'informazione implica una forma semplificata di processo decisionale sequenziale, semplificata perché le osservazioni influiscono soltanto sullo **stato-credenza** dell'agente e non sullo stato fisico esterno. Il valore di ogni particolare osservazione deve derivare dal potenziale di influire sull'azione fisica dell'agente, e tale potenziale può essere stimato direttamente dal modello di decisione.

16.6.1 Un semplice esempio

Supponiamo che una compagnia petrolifera desideri acquisire uno tra n blocchi indistinguibili di diritti di trivellazione oceanica. Supponiamo anche che esattamente uno dei blocchi

contenga petrolio che genererà un utile netto di C dollari, mentre gli altri siano privi di valore. Il prezzo richiesto per ogni blocco è di C/n dollari. Se la compagnia è neutrale rispetto al rischio, le sarà indifferente acquistare un blocco o no, dato che l'utile atteso è zero in entrambi i casi.

Ora immaginiamo che un sismologo offra alla compagnia di effettuare un esame del blocco numero 3, che può indicare con certezza se esso contiene petrolio o no. Quanto dovrebbe essere disposta a pagare la compagnia per tale informazione? Per rispondere a questa domanda occorre esaminare l'uso che ne farebbe.

- Con probabilità $1/n$, l'esame indicherà che il blocco 3 contiene petrolio. In questo caso la compagnia lo acquisterà per C/n dollari ottenendo così un profitto di $C - C/n = (n - 1)C/n$ dollari.
- Con probabilità $(n - 1)/n$, l'esame mostrerà che il blocco non contiene petrolio, nel qual caso la compagnia ne acquisterà un altro. Ora la probabilità di trovare petrolio passa da $1/n$ a $1/(n - 1)$, per cui il profitto atteso della compagnia diventa $C/(n - 1) - C/n = C/n(n - 1)$ dollari.

Ora calcoliamo il profitto atteso dato l'accesso all'informazione del sismologo:

$$\frac{1}{n} \times \frac{(n - 1)C}{n} + \frac{n - 1}{n} \times \frac{C}{n(n - 1)} = C/n.$$

Quindi l'informazione vale C/n dollari per la compagnia, che dovrebbe essere disposta a pagare al sismologo una frazione significativa di questo importo.

Il valore di un'informazione deriva dal fatto che, *con essa*, le proprie azioni possono essere modificate per adattarsi alla situazione *reale*, laddove in sua assenza si può solo scegliere in base alla media di tutte le situazioni possibili. In generale, il valore di una particolare informazione è definito come la differenza nel valore atteso delle azioni migliori prima e dopo la sua acquisizione.

16.6.2 Una formula generale per l'informazione perfetta

È facile derivare una formula matematica generale per esprimere il valore dell'informazione. Ipotizziamo di poter ottenere una evidenza esatta riguardo al valore di qualche variabile casuale E_j (ovvero di poter apprendere $E_j = e_j$) per cui usiamo l'espressione **valore dell'informazione perfetta** (VPI, *value of perfect information*).⁸

Nello stato di informazione iniziale dell'agente, il valore dell'azione migliore corrente α è, dall'Equazione (16.1):

$$EU(\alpha) = \max_a \sum_{s'} P(\text{RISULTATO}(a) = s') U(s'),$$

e il valore della nuova azione migliore (dopo aver acquisito la nuova evidenza $E_j = e_j$) sarà:

$$EU(\alpha_{e_j} | e_j) = \max_a \sum_{s'} P(\text{RISULTATO}(a) = s' | e_j) U(s').$$

**valore
dell'informazione
perfetta**

⁸ Non c'è perdita di espressività nel richiedere informazione perfetta. Supponiamo di voler modellare il caso in cui acquisiamo un maggior grado di certezza su una variabile. Possiamo farlo introducendo un'altra variabile per cui apprendiamo informazione perfetta. Per esempio, supponiamo di avere inizialmente grande incertezza sulla variabile *Temperatura*. Poi otteniamo la conoscenza perfetta che *Termometro* = 37; questo ci fornisce informazione imperfetta sulla vera *Temperatura*, e l'incertezza dovuta all'errore di misura è codificata nel modello sensoriale $\mathbf{P}(\text{Termometro} | \text{Temperatura})$. Cfr. l'Esercizio 16.VPIX per un altro esempio.

Ma E_j è una variabile casuale il cui valore è *attualmente* sconosciuto, perciò dobbiamo fare la media su tutti i possibili valori e_{jk} di E_j usando le nostre credenze *correnti* sul suo valore, per determinare il valore di scoprire E_j :

$$VPI(E_j) = \left(\sum_{e_j} P(E_j = e_j) EU(\alpha_{e_j} | E_j = e_j) \right) - EU(\alpha).$$

Per comprendere intuitivamente questa formula, considerate il semplice caso in cui ci sono solo due azioni tra cui scegliere, a_1 e a_2 , le cui utilità attese correnti sono U_1 e U_2 . L'informazione $E_j = e_j$ darà le nuove utilità attese U'_1 e U'_2 , ma prima di ottenere E_j avremo delle distribuzioni di probabilità sui possibili valori di U'_1 e U'_2 (che ipotizziamo siano indipendenti).

Supponiamo che a_1 e a_2 rappresentino due possibili tragitti per l'attraversamento di un massiccio montuoso in inverno. a_1 è una bella autostrada dritta che sfrutta una galleria, mentre a_2 è una strada sterrata che si inerpica tortuosamente sulle cime. Avendo a disposizione solo questa informazione, il tragitto a_1 è palesemente preferibile, perché è possibile che il tragitto a_2 sia bloccato da valanghe, mentre è abbastanza improbabile che in a_1 ci siano dei blocchi. Ne consegue che l'utilità U_1 è chiaramente più alta di U_2 . È anche possibile ottenere fotografie satellitari E_j sullo stato effettivo di ogni strada; queste ci potrebbero dare nuove utilità attese U'_1 e U'_2 . Le distribuzioni di queste ultime sono riportate nella Figura 16.8(a). È ovvio che in questo caso non vale la pena di acquistare le fotografie satellitari, perché è molto improbabile che l'informazione ottenuta da esse risulti in un cambiamento nei nostri piani. Senza cambiamento, l'informazione non ha valore.

Ora supponiamo di dover scegliere tra due strade sterrate tortuose di lunghezza leggermente diversa e di dover trasportare un passeggero gravemente ferito. In questo caso, anche se U_1 e U_2 sono molto vicine, le distribuzioni di U'_1 e U'_2 sono molto ampie. C'è una possibilità significativa che il secondo tragitto sia libero mentre il primo è bloccato, nel qual caso la differenza tra le utilità sarà molto alta. La formula del VPI indica che potrebbe valere la pena di pagare per ottenere le foto satellitari. Questa situazione è illustrata dalla Figura 16.8(b).

Infine, supponiamo di dover scegliere tra due strade sterrate in estate, quando la possibilità di blocchi dovuti alla neve è quasi assente. In questo caso le foto satellitari potrebbero mostrare che una strada è più piacevole dell'altra a causa della fioritura dei prati alpini, o forse più scomoda a causa di piogge recenti. È quindi abbastanza probabile che l'acquisizione dell'informazione potrebbe causare una modifica del nostro piano: in questo caso, tuttavia, è anche probabile che la differenza di valore tra le due strade rimanga abbastanza piccola, per cui non vale la pena di ottenere le foto. Questa situazione è illustrata dalla Figura 16.8(c).

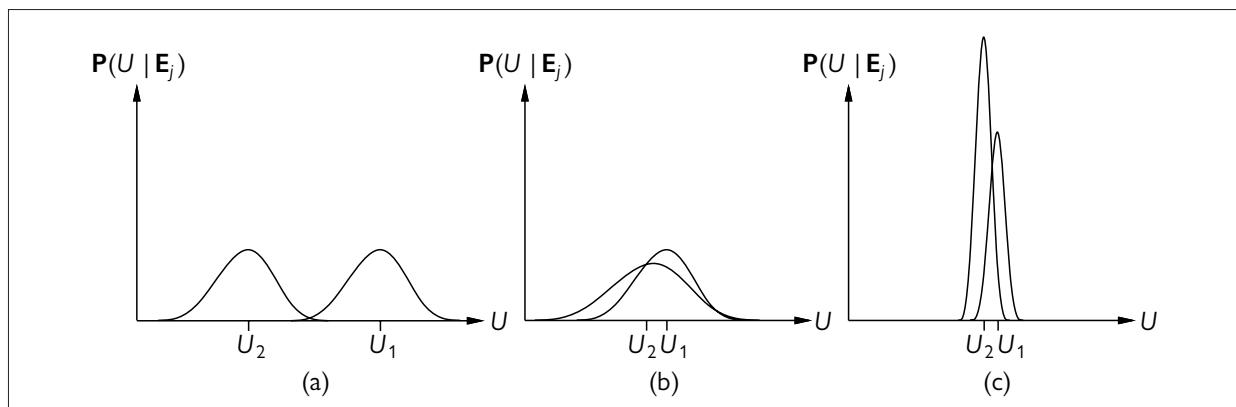


Figura 16.8 Tre casi generici del valore dell'informazione. In (a) a_1 rimarrà quasi certamente superiore ad a_2 , per cui l'informazione è superflua. In (b) la scelta non è chiara, e l'informazione risulta cruciale. In (c) la scelta non è chiara ma, dal momento che non c'è molta differenza, l'informazione ha meno valore. Nota: il fatto che U'_2 abbia un picco più alto in (c) significa che il suo valore atteso è noto con certezza maggiore rispetto a U'_1 .

In definitiva, l'informazione ha un valore nella misura in cui è probabile che sia causa di un cambiamento nel piano e nella misura in cui il nuovo piano sarà significativamente migliore del precedente.



16.6.3 Proprietà del valore dell'informazione

Ci si potrebbe chiedere se è possibile che l'informazione possa rivelarsi deleteria, ovvero che abbia valore negativo. Intuitivamente, ci aspettiamo che ciò sia impossibile: dopotutto, nel caso peggiore potremmo sempre ignorarla facendo finta di non averla mai ricevuta. Questo è confermato dal seguente teorema, che si applica a qualsiasi agente basato sulla teoria delle decisioni che usi qualsiasi rete di decisione con possibili osservazioni E_j :

Il valore atteso dell'informazione è non negativo:

$$\forall_j, VPI(E_j) \geq 0.$$



Il teorema segue direttamente dalla definizione di VPI: lasciamo la dimostrazione al lettore (cfr. Esercizio 16.NNVP). Naturalmente si tratta di un teorema che fa riferimento a un valore *atteso* e non *attuale*. La disponibilità di informazioni aggiuntive può facilmente portare a un piano che *risulta* peggiore dell'originale, in presenza di informazioni fuorvianti. Per esempio, un esame medico che fornisce un risultato falso positivo può portare a un intervento chirurgico non necessario, ma questo non significa che l'esame non andava fatto.

È importante ricordare che il VPI dipende dallo stato corrente dell'informazione, e può cambiare a mano a mano che si acquisiscono ulteriori dati. Per ogni evidenza E_j , il valore della sua acquisizione può essere discendente (per esempio se un'altra variabile pone un vincolo forte sulla distribuzione a posteriori per E_j) o ascendente (per esempio se un'altra variabile fornisce un indizio sfruttabile conoscendo E_j , in modo da consentire di mettere a punto un piano nuovo e migliore). Ne consegue che il VPI non è una quantità additiva, vale a dire:

$$VPI(E_j, E_k) \neq VPI(E_j) + VPI(E_k) \quad (\text{in generale}).$$

VPI è comunque indipendente dall'ordine. Quindi:

$$VPI(E_j, E_k) = VPI(E_j) + VPI(E_k|E_j) = VPI(E_k) + VPI(E_j|E_k) = VPI(E_k, E_j).$$

dove la notazione $VPI(\bullet|E)$ indica il VPI calcolato secondo la distribuzione a posteriori in cui E è già stata osservata. L'indipendenza dall'ordine contraddistingue le azioni di percezione da quelle ordinarie e semplifica il problema di calcolare il valore di una sequenza di azioni percettive. Torneremo su questo punto nel prossimo paragrafo.

16.6.4 Implementazione di un agente che raccoglie informazioni

Un agente assennato dovrebbe porre domande in un ordine ragionevole, evitare di domandare cose irrilevanti, prendere in considerazione l'importanza di ogni informazione in rapporto al suo costo e smettere di domandare nel momento opportuno. È possibile ottenere tutte queste capacità usando come guida il valore dell'informazione.

La Figura 16.9 riporta il progetto generale di un agente in grado di raccogliere informazioni in modo intelligente prima di agire. Per ora assumeremo che a ogni variabile di evidenza osservabile E_j sia associato un $\text{Costo}(E_j)$, che riflette appunto il costo di acquisizione di tale variabile attraverso esami, consulenze, domande, eccetera. L'agente richiede quella che gli sembra l'osservazione più efficiente in termini di guadagno di utilità per unità di costo. Ipotizziamo che come risultato dell'azione $\text{Richiedi}(E_j)$, la percezione successiva fornisca il valore di E_j . Se nessuna osservazione vale il suo costo, l'agente seleziona una azione "vera".

L'algoritmo descritto implementa una forma di acquisizione di informazione detta **miope** perché sfrutta la formula del VPI senza "guardare in là", calcolando sempre il valore dell'informazione come se stesse acquisendo una sola variabile di evidenza. Il controllo miope è basato sulla stessa idea euristica della ricerca greedy, e spesso funziona bene nella pratica

miope

```

function AGENTE-RACCOLTORE-DI-INFORMAZIONI(percezione) returns un’azione
persistent: D, una rete di decisione

    integra percezione in D
    j  $\leftarrow$  il valore che massimizza  $VPI(E_j) / C(E_j)$ 
    if  $VPI(E_j) > C(E_j)$ 
        then return Richiedi(Ej)
    else return la migliore azione in base a D

```

Figura 16.9 Progetto di un semplice agente miope in grado di raccogliere informazioni. L’agente opera selezionando ripetutamente l’osservazione con il più alto valore dell’informazione, finché il costo dell’osservazione successiva supera il beneficio atteso.

(per esempio, si è dimostrato più bravo di molti medici esperti nel selezionare gli esami diagnostici da effettuare). Tuttavia, se non c’è una singola variabile di evidenza che possa fornire un aiuto significativo, un agente miope potrebbe intraprendere frettolosamente un’azione quando invece sarebbe stato meglio richiedere prima due o più variabili. Nel paragrafo seguente consideriamo la possibilità di ottenere osservazioni multiple.

16.6.5 Raccolta di informazioni non miope

Il fatto che il valore di una sequenza di osservazioni non cambi con permutazioni della sequenza è interessante, ma di per sé non conduce ad algoritmi efficienti per una raccolta di informazioni ottima. Anche se ci limitiamo a scegliere in anticipo un sottoinsieme fissato di osservazioni da raccogliere, ci sono 2^n possibili sottoinsiemi per n potenziali osservazioni. Nel caso generale dobbiamo affrontare un problema ancora più complesso, quello di trovare un *piano condizionale* (cfr. Paragrafo 11.5.2) che scelga un’osservazione e agisca o scelga altre osservazioni, in base al risultato. Piani come questi formano degli alberi il cui numero è superesponenziale in n .⁹

caccia al tesoro

Per osservazioni di variabili in una rete di decisione, questo problema risulta intrattabile anche quando la rete è un polialbero. Esistono però casi particolari in cui il problema può essere risolto in modo efficiente; uno di questi lo presentiamo qui: il problema della **caccia al tesoro** (o della **sequenza di test di costo minimo**, per chi non ama i romanticismi). Ci sono n posizioni $1, \dots, n$; ogni posizione i contiene un tesoro con probabilità indipendente $P(i)$; verificare la posizione i ha un costo $C(i)$. Il problema corrisponde a una rete di decisione in cui tutte le potenziali variabili di evidenza $Tesoro_i$ sono assolutamente indipendenti. L’agente esamina le posizioni in un certo ordine finché trova un tesoro. La domanda è: qual è l’ordine ottimo?

Per rispondere dobbiamo considerare i costi attesi e le probabilità di successo di varie sequenze di osservazioni, ipotizzando che l’agente si fermi quando trova il tesoro. Sia \mathbf{x} una tale sequenza; \mathbf{xy} la concatenazione delle sequenze \mathbf{x} e \mathbf{y} ; $C(\mathbf{x})$ il costo atteso di \mathbf{x} ; $P(\mathbf{x})$ la probabilità che la sequenza \mathbf{x} riesca a trovare il tesoro; $F(\mathbf{x}) = 1 - P(\mathbf{x})$ la probabilità che fallisca. Date queste definizioni, abbiamo:

$$C(\mathbf{xy}) = C(\mathbf{x}) + F(\mathbf{x})C(\mathbf{y}), \quad (16.3)$$

il che significa che la sequenza \mathbf{xy} sostiene il costo di \mathbf{x} e, se \mathbf{x} fallisce, anche quello di \mathbf{y} .

⁹ Il problema generale di generare comportamenti sequenziali in un ambiente parzialmente osservabile ricade nella categoria dei **processi decisionali di Markov parzialmente osservabili**, descritti nel Capitolo 17.

L'idea di base, in qualsiasi problema di ottimizzazione di sequenze, è di osservare la variazione di costo, definita da $\Delta = C(\mathbf{wxyz}) - C(\mathbf{wyxz})$, che si ha quando sue sottosequenze adiacenti \mathbf{x} e \mathbf{y} in una sequenza generale \mathbf{wxyz} sono scambiate di posto. Nel caso di una sequenza ottima, variazioni come queste peggiorano il costo. Il primo passo consiste nel mostrare che il segno dell'effetto (aumento o riduzione del costo) non dipende dal contesto fornito da \mathbf{w} e \mathbf{z} . We Abbiamo:

$$\begin{aligned}\Delta &= [C(\mathbf{w}) + F(\mathbf{w})C(\mathbf{xyz})] - [C(\mathbf{w}) + F(\mathbf{w})C(\mathbf{yxz})] \quad (\text{per l'Equazione (16.3)}) \\ &= F(\mathbf{w})[C(\mathbf{xyz}) - C(\mathbf{yxz})] \\ &= F(\mathbf{w})[(C(\mathbf{xy}) + F(\mathbf{xy})C(\mathbf{z})) - (C(\mathbf{yx}) + F(\mathbf{yx})C(\mathbf{z}))] \quad (\text{per l'Equazione (16.3)}) \\ &= F(\mathbf{w})[C(\mathbf{xy}) - C(\mathbf{yx})] \quad (\text{poiché } F(\mathbf{xy}) = F(\mathbf{yx})).\end{aligned}$$

Abbiamo così mostrato che la direzione della variazione di costo per l'intera sequenza dipende solo dalla direzione della variazione di costo della coppia di elementi che vengono scambiati; il contesto invece non conta. Questo ci consente di ordinare la sequenza mediante confronti di coppie per ottenere una soluzione ottima. Nello specifico, ora abbiamo:

$$\begin{aligned}\Delta &= F(\mathbf{w})[(C(\mathbf{x}) + F(\mathbf{x})C(\mathbf{y})) - (C(\mathbf{y}) + F(\mathbf{y})C(\mathbf{x}))] \quad (\text{per l'Equazione (16.3)}) \\ &= F(\mathbf{w})[C(\mathbf{x})(1 - F(\mathbf{y})) - C(\mathbf{y})(1 - F(\mathbf{x}))] = F(\mathbf{w})[C(\mathbf{x})P(\mathbf{y}) - C(\mathbf{y})P(\mathbf{x})].\end{aligned}$$

Questo vale per qualsiasi sequenza \mathbf{x} e \mathbf{y} , perciò vale nello specifico quando \mathbf{x} e \mathbf{y} sono singole osservazioni di posizioni i e j , rispettivamente. Sappiamo dunque che, perché i e j siano adiacenti in una sequenza ottima, dobbiamo avere $C(i)P(j) \leq C(j)P(i)$, o $\frac{P(i)}{C(i)} \geq \frac{P(j)}{C(j)}$. In altre parole, l'ordine ottimo classifica le posizioni in base alla probabilità di successo per costo unitario. L'Esercizio 16.HUNT chiede di determinare se questa sia effettivamente la strategia seguita dall'algoritmo della Figura 16.9 per questo problema.

16.6.6 Analisi di sensibilità e decisioni robuste

L'**analisi di sensibilità** è una pratica ampiamente diffusa nelle discipline tecnologiche: consiste nell'analizzare quanto cambia l'output di un processo se si variano i parametri del modello. Nei sistemi probabilistici e di teoria delle decisioni l'analisi di sensibilità è particolarmente importante perché le probabilità sono generalmente apprese dai dati o stimate da esperti umani, il che significa che sono anch'esse soggette a un notevole grado di incertezza. Soltanto in casi rari, come il lancio dei dadi nel backgammon, le probabilità sono oggettivamente note.

analisi di sensibilità

Nel caso di un processo decisionale guidato dall'utilità, si può vedere l'output come la decisione effettivamente presa o l'utilità attesa di tale decisione. Consideriamo prima la seconda: poiché l'utilità attesa dipende dalle probabilità considerate nel modello, possiamo calcolare la derivata dell'utilità attesa di qualsiasi azione data rispetto a ognuno di quei valori di probabilità (per esempio, se tutte le distribuzioni di probabilità condizionate nel modello sono riportate esplicitamente in una tabella, il calcolo dei valori attesi richiede di calcolare il rapporto di due somme di prodotti; per ulteriori dettagli cfr. il Capitolo 20 del Volume 2). Possiamo quindi determinare quali parametri del modello abbiano il maggiore effetto sull'utilità attesa della decisione finale.

Se invece ci interessa la decisione effettivamente presa, più che la sua utilità secondo il modello, possiamo semplicemente variare i parametri in modo sistematico (magari usando una ricerca binaria) per verificare se la decisione cambia, e in questo caso, qual è la più piccola perturbazione che causa tale cambiamento. Si potrebbe pensare che non conti tanto quale decisione venga presa, ma solo la sua utilità. In effetti è così, ma nella pratica potrebbe esserci una differenza veramente notevole tra l'utilità *reale* di una decisione e l'utilità *secondo il modello*.

Se tutte le perturbazioni ragionevoli dei parametri lasciano invariata la decisione ottima, è ragionevole ipotizzare che la decisione sia buona, anche se la stima di utilità per tale deci-

decisione robusta

sione non è del tutto corretta. D'altra parte, se la decisione ottima cambia notevolmente al variare dei parametri del modello, c'è una buona possibilità che il modello possa produrre una decisione che in realtà è subottima. In tal caso, vale la pena di impegnarsi di più per raffinare il modello.

Questi punti intuitivi sono stati formalizzati in diversi campi (teoria del controllo, analisi decisionale, gestione dei rischi) che propongono il concetto di decisione **robusta** o **minimax**, nel senso di una decisione che fornisce il miglior risultato nel caso peggiore. In questo caso “caso peggiore” significa rispetto a tutte le variazioni plausibili dei parametri del modello. Se indichiamo con θ tutti i parametri del modello, la decisione robusta è definita da:

$$a^* = \operatorname{argmax}_a \min_{\theta} EU(a; \theta).$$

In molti casi, particolarmente nella teoria del controllo, l'approccio della decisione robusta porta a progetti che funzionano in modo molto affidabile nella pratica; in altri casi, invece, porta a decisioni eccessivamente prudenti. Per esempio, nel progetto di un'auto a guida autonoma, l'approccio robusto ipotizzerebbe il caso peggiore per il comportamento degli altri veicoli sulle strade – ritenendoli guidati tutti da maniaci omicidi. In tal caso, la soluzione ottima per l'auto è quella di rimanere nel box.

La teoria decisionale bayesiana offre un'alternativa ai metodi robusti: se vi è incertezza sui parametri del modello, tale incertezza va modellata usando iperparametri.

Mentre l'approccio robusto affermerebbe che qualche probabilità θ_i nel modello ha possibili valori compresi tra 0,3 e 0,7, con il valore effettivo scelto da un avversario per fare andare le cose nel modo peggiore possibile; l'approccio bayesiano assegnerebbe una distribuzione di probabilità a priori su θ_i e proseguirebbe come prima. Questo approccio richiede un maggiore lavoro di modellazione – per esempio, il modellista bayesiano deve decidere se i parametri θ_i e θ_j sono indipendenti – ma spesso consente di ottenere prestazioni migliori nella pratica.

Oltre all'incertezza dei parametri, le applicazioni della teoria delle decisioni nel mondo reale soffrono anche da incertezza *strutturale*. Per esempio, l'assunto di indipendenza di *TrafficoAereo*, *ControversieLegali* e *Costruzione* nella Figura 16.6 potrebbe essere errato, e potrebbero esistere variabili aggiuntive che il modello semplicemente omette. Al momento non sappiamo bene come tenere conto di questo tipo di incertezza. Una possibilità è quella di mantenere un insieme di modelli, magari generati da algoritmi di apprendimento automatico, nella speranza di catturare così le variazioni significative.

16.7 Preferenze ignote

In questo paragrafo vediamo che cosa succede quando vi è incertezza sulla funzione di utilità di cui si vuole ottimizzare il valore atteso. Esistono due versioni di questo problema: una in cui un agente (macchina o essere umano) è incerto sulla *sua stessa* funzione di utilità e un'altra in cui una macchina che dovrebbe aiutare un essere umano è incerta sui desideri di quest'ultimo.

16.7.1 Incertezza sulle proprie preferenze

Immaginate di trovarvi in una gelateria in Tailandia e che siano rimasti soltanto due gusti di gelato: vaniglia e durian. Entrambi costano 2 dollari. Sapete che il gusto di vaniglia vi piace e sareste disponibili a pagare fino a 3 dollari per un gelato alla vaniglia in una giornata così calda, perciò scegliendo il vaniglia c'è un guadagno netto di 1 dollaro. D'altra parte, non avete idea se il gelato al durian vi piaccia o no, ma avete letto su Wikipedia che durian suscita

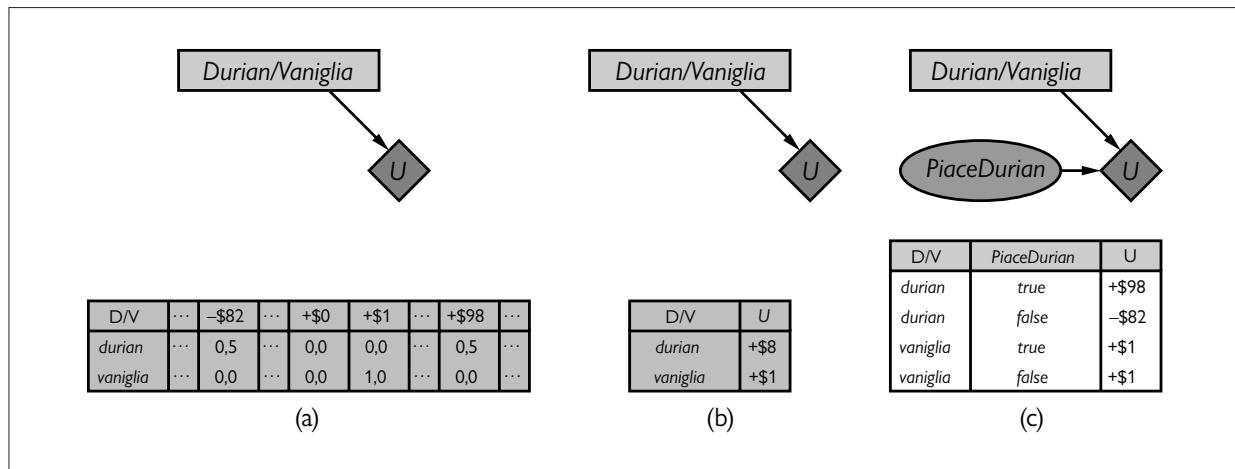


Figura 16.10 (a) Una rete di decisione per la scelta del gelato con una funzione di utilità incerta. (b) La rete con l'utilità attesa di ogni azione. (c) L'incertezza viene spostata dalla funzione di utilità a una nuova variabile casuale.

reazioni diverse a persone differenti: alcuni lo trovano “il gusto alla frutta migliore del mondo” mentre altri lo considerano “una schifezza, da vomito, puzzolente e nauseabondo”.

Per dare qualche numero, diciamo che c’è una probabilità del 50% che lo troverete sublime (+\$100) e del 50% che lo odierete (-\$80 se il sapore rimane in bocca per tutto il pomeriggio). In questo caso non c’è incertezza su quale premio vincerete – è sempre lo stesso gelato al durian – ma c’è incertezza sulle vostre stesse preferenze per tale premio.

Potremmo estendere il formalismo delle reti di decisione in modo da consentire le utilità incerte, come mostrato nella Figura 16.10(a). Tuttavia, se non è possibile acquisire ulteriori informazioni sulle vostre preferenze riguardo il durian – per esempio, se la gelateria non vi lascerà fare prima un assaggio – il problema di decisione è identico a quello mostrato nella Figura 16.10(b). Possiamo semplicemente sostituire il valore incerto del durian con il suo guadagno netto atteso di $(0,5 \times \$100) - (0,5 \times \$80) - \$2 = \8 e la vostra decisione sarà invariata.

Se c’è la possibilità che la vostra opinione riguardo il durian cambi, magari perché fate un assaggio, o venite a sapere che tutti i vostri parenti amano il durian, allora la trasformazione della Figura 16.10(b) non è valida. Tuttavia, possiamo comunque trovare un modello equivalente in cui la funzione di utilità è deterministica. Anziché dire che vi è incertezza sulla funzione di utilità, trasferiamo tale incertezza “nel mondo”, per così dire. In pratica, creiamo una nuova variabile casuale, *PiaceDurian* con probabilità a priori di 0,5 per *true* e *false*, come illustrato nella Figura 16.10(c). Con questa variabile in più, la funzione di utilità diventa deterministica, ma possiamo comunque gestire una variazione delle preferenze per il durian.

Il fatto che sia possibile modellare preferenze ignote per mezzo di variabili casuali ordinarie significa che possiamo continuare a usare gli apparati e i teoremi sviluppati per le preferenze note. D’altra parte, questo non significa che possiamo sempre assumere che le preferenze siano note. L’incertezza esiste e influisce su come dovrebbero comportarsi gli agenti.

16.7.2 Delega agli esseri umani

Passiamo ora al secondo dei casi citati in precedenza: quello di una macchina che dovrebbe aiutare un essere umano, ma è incerta su ciò che l’essere umano voglia. Dobbiamo rimandare la trattazione completa del caso al Capitolo 18, dove esamineremo le decisioni che coinvolgono più di un agente. Qui ci poniamo una sola, semplice domanda: in quali circostanze una simile macchina si affiderà all’essere umano?

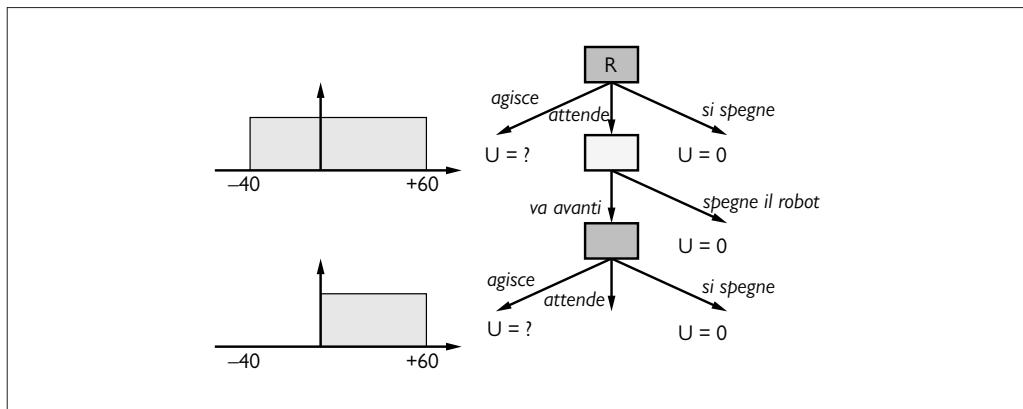


Figura 16.11 Il gioco del pulsante di spegnimento. R , il robot, può scegliere di agire ora, con un payoff molto incerto; di spegnersi; oppure di delegare a H , l’essere umano. H può spegnere R oppure lasciare che vada avanti. R ora ha di nuovo la stessa scelta. Agire ha ancora un payoff incerto, ma ora R sa che il payoff non è negativo.

Per studiare il problema, consideriamo uno scenario molto semplice, mostrato nella Figura 16.11. Robbie è un robot software che lavora come assistente personale per Harriet, un essere umano molto impegnato. Harriet ha bisogno di una stanza d’albergo per il suo prossimo incontro d’affari a Ginevra. Robbie può agire ora – per esempio può prenotare per Harriet un albergo molto costoso vicino al luogo previsto per la riunione d’affari, ma non sa quanto Harriet gradirà quell’albergo e il prezzo della stanza; diciamo che ha una probabilità uniforme per il suo valore netto per Harriet, tra -40 e $+60$, con una media di $+10$. Robbie potrebbe anche “spegnersi da solo” – o per dirla in termini meno melodrammatici, tirarsi fuori dal processo di prenotazione dell’albergo – caso per cui definiamo (senza perdita di generalità) un valore 0 per Harriet. Se le sue due scelte fossero quelle, Robbie andrebbe avanti e prenoterebbe l’albergo, correndo un rischio significativo di non soddisfare Harriet (se l’intervallo fosse da -60 a $+40$, con media di -10 , Robbie si spegnerebbe da solo). Ma noi diamo a Robbie una terza scelta: spiegare il suo piano, attendere e lasciare che sia Harriet a spegnerlo. Harriet può quindi spegnere il robot Robbie oppure lasciarlo proseguire e prenotare l’albergo. Quale vantaggio potrebbe portare questo approccio, potremmo chiederci, dato che Harriet avrebbe potuto fare entrambe le scelte da sola?

Il punto è che la scelta di Harriet – spegnere Robbie o lasciarlo proseguire – fornisce a Robbie informazioni sulle preferenze di Harriet. Ipotizzeremo, per ora, che Harriet sia razionale, perciò, se lascia proseguire Robbie, significa che tale azione ha un valore positivo per Harriet. Ora, come è mostrato nella Figura 16.11, la credenza di Robbie cambia ed è uniforme tra 0 e $+60$, con media di $+30$.

Quindi, se valutiamo le scelte iniziali di Robbie dal suo punto di vista:

1. Agire ora e prenotare l’albergo ha un valore atteso di $+10$.
2. Spegnersi da solo ha un valore di 0 .
3. Attendere e lasciare che sia Harriet a spegnerlo porta a due possibili esiti:
 - c’è una probabilità del 40% , sulla base dell’incertezza di Robbie riguardo le preferenze di Harriet, che Harriet non gradirà il piano e spegnerà Robbie, con valore 0 ;
 - c’è una probabilità del 60% che Harriet gradirà il piano e consentirà a Robbie di proseguire, con valore atteso $+30$.

Quindi, l’attesa ha un valore atteso di $(0,4 \times 0) + (0,6 \times 30) = +18$, migliore del $+10$ che Robbie si aspetta nel caso in cui agisca ora.

Il risultato è che Robbie ha un incentivo positivo ad affidarsi a Harriet, cioè a consentire di venire spento. Questo incentivo deriva direttamente dall'incertezza di Robbie riguardo le preferenze di Harriet. Robbie è consapevole che c'è una probabilità (del 40% in questo esempio) che faccia qualcosa che Harriet non gradirà, con la conseguenza che essere spento sarebbe preferibile a proseguire. Se Robbie fosse già certo delle preferenze di Harriet, andrebbe avanti e prenderebbe la decisione (o si spagnerebbe); rivolgersi a Harriet non fornirebbe assolutamente alcun guadagno, poiché, secondo le credenze di Robbie, egli è già in grado di predire con esattezza ciò che Harriet deciderà.

In effetti, è possibile dimostrare lo stesso risultato nel caso generale: finché Robbie non è completamente certo che sta per fare ciò che Harriet stessa farebbe, è meglio se consente a lei di spegnerlo. Intuitivamente, la decisione di Harriet fornisce a Robbie delle informazioni, e il valore atteso delle informazioni è sempre non negativo. Viceversa, se Robbie è certo della decisione di Harriet, la sua decisione non fornisce nuove informazioni, perciò Robbie non ha alcun incentivo a lasciar decidere Harriet.

Formalmente, sia $P(u)$ la densità di probabilità a priori di Robbie sull'utilità di Harriet per l'azione proposta a . Allora il valore di proseguire con l'azione a è:

$$EU(a) = \int_{-\infty}^{\infty} P(u) \cdot u du = \int_{-\infty}^0 P(u) \cdot u du + \int_0^{\infty} P(u) \cdot u du.$$

(Vedremo tra breve il motivo per cui l'integrale è suddiviso in questo modo). D'altra parte, il valore dell'azione d – delegare a Harriet – si compone di due parti: se $u > 0$ allora Harriet consente a Robbie di proseguire, perciò il valore è u , ma se $u < 0$ allora Harriet spegne Robbie, perciò il valore è 0:

$$EU(d) = \int_{-\infty}^0 P(u) \cdot 0 du + \int_0^{\infty} P(u) \cdot u du.$$

Confrontando le espressioni per $EU(a)$ e $EU(d)$, vediamo subito che:

$$EU(d) \geq EU(a)$$

poiché nell'espressione per $EU(d)$ la regione di utilità negativa è azzerata. Le due scelte hanno pari valore soltanto quando la regione di utilità negativa ha probabilità zero, cioè quando Robbie è già certo che Harriet apprezzi l'azione proposta.

Questo modello si presta ad alcune elaborazioni immediate che è utile esaminare subito. La prima consiste nell'imporre un costo per il tempo di Harriet. In tal caso, Robbie è meno incentivato a disturbare Harriet se il rischio di sbagliare è basso. Ed è giusto così. Inoltre, se Harriet non ama proprio essere interrotta, non dovrebbe essere troppo sorpresa se Robbie talvolta esegue azioni che a lei non piacciono.

La seconda elaborazione consiste nel consentire una certa probabilità di errore umano: Harriet potrebbe a volte spegnere Robbie quando l'azione proposta è ragionevole, e a volte potrebbe lasciarlo proseguire quando l'azione proposta non è desiderabile. È facile introdurre questa probabilità di errore nel modello (cfr. l'Esercizio 16.OFF). Come è facile prevedere, la soluzione mostra che Robbie è meno incline a rivolgersi a una Harriet irrazionale che a volte agisce contro il suo stesso interesse. Maggiore è la casualità del comportamento di Harriet, più incerto deve essere Robbie riguardo le sue preferenze, prima di rivolgersi a lei. Anche qui possiamo dire che è giusto che le cose stiano così: per esempio, se Robbie è un'automobile a guida autonoma e Harriet è il suo terribile passeggero di due anni, Robbie non dovrebbe consentire a Harriet di spegnerlo mentre sta percorrendo un'autostrada.

16.8 Riepilogo

Questo capitolo ha mostrato come combinare la teoria dell'utilità con la probabilità per permettere a un agente di scegliere le azioni che massimizzeranno le sue prestazioni attese.

- La **teoria della probabilità** descrive le credenze che un agente dovrebbe avere sulla base delle evidenze, la **teoria dell'utilità** descrive i suoi desideri, e la **teoria delle decisioni** mette insieme le due cose per descrivere ciò che l'agente deve fare.
- Possiamo usare la teoria delle decisioni per costruire un sistema che prende decisioni considerando tutte le azioni possibili e scegliendo quella che porta all'esito atteso più favorevole. Un sistema simile è chiamato **agente razionale**.
- La teoria dell'utilità mostra che un agente le cui preferenze tra lotterie sono consistenti con un insieme di semplici assiomi può essere descritto come se avesse una funzione di utilità; inoltre l'agente sceglierà le azioni che massimizzano la sua utilità attesa.
- La **teoria dell'utilità multiattributo** si occupa delle utilità che dipendono da più attributi distinti degli stati. La **dominanza stocastica** è una tecnica particolarmente utile per prendere decisioni non ambigue, anche in assenza di valori precisi di utilità degli attributi.
- Le **reti di decisione** forniscono un semplice formalismo per esprimere e risolvere problemi decisionali. Sono un'estensione naturale delle reti bayesiane e, oltre ai nodi di causalità, contengono nodi di decisione e nodi di utilità.
- Talvolta, per risolvere un problema è meglio raccogliere nuove informazioni prima di prendere una decisione. Il **valore dell'informazione** è definito come il miglioramento atteso nell'utilità della decisione rispetto a quella presa in assenza di tale informazione. È particolarmente utile per guidare il processo di raccolta delle informazioni prima di prendere una decisione finale.
- Quando, come avviene spesso, è impossibile specificare la funzione di utilità dell'essere umano in modo completo e corretto, le macchine devono operare in condizioni di incertezza circa il vero obiettivo. Questo fa la differenza quando la macchina ha la possibilità di acquisire ulteriori informazioni sulle preferenze umane. Abbiamo visto, con un'argomentazione semplice, che l'incertezza sulle preferenze garantisce che la macchina deleghi all'essere umano, fino al punto di lasciarsi spegnere.

Note storiche e bibliografiche

Nel trattato del XVII secolo *L'art de Penser*, o *Port-Royal Logic*, Arnauld (1662) afferma:

Per giudicare che cosa si dovrebbe fare per ottenere un bene o evitare un male, è necessario considerare non solo il bene e il male di per sé, ma anche la probabilità che si verifichi o non si verifichi; e visualizzare geometricamente la proporzione di tutti questi elementi.

I testi moderni parlano di *utilità* anziché di bene e male, ma la frase precedente afferma, correttamente, che occorre moltiplicare l'utilità per la probabilità (“visualizzare geometricamente”) per ottenere l'utilità attesa, e massimizzare quest'ultima su tutti i risultati (“tutti questi elementi”) per “giudicare che cosa si deve fare”. È notevole come Arnauld avesse visto giusto, più di 350 anni fa e soltanto 8 anni dopo che Pascal e

Fermat mostraronono per primi come utilizzare correttamente la probabilità.

Daniel Bernoulli (1738), investigando il paradosso di San Pietroburgo (cfr. Esercizio 16.STPT) fu il primo a capire l'importanza delle misure di preferenza applicate alle lotterie, scrivendo “il *valore* di un oggetto non dev'essere basato sul suo *prezzo*, ma piuttosto sull'*utilità* che esso apporta” (corsivo suo). Il filosofo utilitarista Jeremy Bentham (1823) ha proposto l'adozione del **calcolo edonistico** per misurare “piaceri” e “dolori”, sostenendo che tutte le decisioni (e non solo quelle legate al denaro) possono essere ridotte a un confronto tra utilità.

L'introduzione da parte di Bernoulli dell'utilità – una quantità interna, soggettiva – per spiegare il com-

portamento umano attraverso una teoria matematica fu davvero notevole per quei tempi. Ancora più notevole per il fatto che, a differenza degli importi monetari, i valori di utilità di varie puntate e premi non sono direttamente osservabili; le utilità devono essere inferte dalle preferenze mostrate da un individuo. Dovettero passare due secoli perché le implicazioni di tale concetto fossero analizzate appieno e l'utilità divenisse largamente accettata da statistici ed economisti.

La derivazione di utilità numeriche dalle preferenze fu effettuata per la prima volta da Ramsey (1931); gli assiomi sulla preferenza presentati in questo testo hanno una forma più vicina a quelli riscoperti in *Teoria dei giochi e comportamento economico* (von Neumann e Morgenstern, 1944). Ramsey aveva derivato probabilità soggettive (e non solo utilità) partendo dalle preferenze di un agente; Savage (1954) e Jeffrey (1983) offrirono presentazioni più recenti di metodologie analoghe. Beardon *et al.* (2002) mostrarono che una funzione di utilità non è sufficiente per rappresentare preferenze non transitive e altre situazioni anomale.

Nel periodo post-bellico, la teoria delle decisioni divenne uno strumento standard nei campi dell'economia, della finanza e del management. Emerse un campo di **analisi decisionale** per aiutare a prendere decisioni più razionali in aree quali la strategia militare, le diagnosi mediche, la salute pubblica, la progettazione tecnica e la gestione di risorse. Il processo coinvolge un **decisore** che stabilisce le preferenze tra i risultati e un **analista decisionale** che enumera le possibili azioni e i possibili risultati e elicita le preferenze del decisore per determinare il miglior corso d'azione. Von Winterfeldt ed Edwards (1986) offrirono una prospettiva sull'analisi delle decisioni e sulle sue sottili relazioni con le strutture di preferenza umane. Smith (1998) fornì una panoramica sulla metodologia dell'analisi decisionale.

Fino agli anni 1980, i problemi di decisioni multivariate erano affrontati costruendo "alberi di decisione" con tutte le possibili istanze delle variabili. Le reti di decisione, o diagrammi di influenza, che sfruttano le stesse proprietà di indipendenza condizionale delle reti bayesiane, sono state introdotte da Howard e Matheson (1984), basandosi sul lavoro svolto precedentemente allo SRI (Miller *et al.*, 1976). L'algoritmo di Howard e Matheson costruiva l'albero di decisione completo (esponenzialmente grande) a partire dalla rete di decisione. Shachter (1986) sviluppò un metodo per prendere decisioni partendo direttamente dalla rete, senza creare un albero. Questo algoritmo fu anche uno dei primi a realizzare l'inferenza completa

per reti bayesiane a connessioni multiple. Nilsson e Lauritzen (2000) collegarono gli algoritmi per le reti di decisione ai correnti sviluppi degli algoritmi di clustering per reti bayesiane. La collezione di Oliver e Smith (1990) contiene diversi utili articoli preliminari sulle reti di decisione, così come il numero speciale del 1990 della rivista *Networks*. Il testo di Fenton e Neil (2018) fornisce un'utile guida per risolvere problemi decisionali del mondo reale utilizzando le reti di decisione. Articoli sulle reti di decisione e sulla modellazione dell'utilità compaiono regolarmente anche nelle riviste *Management Science* e *Decision Analysis*.

Un numero sorprendentemente basso di ricercatori di IA ha adottato strumenti della teoria delle decisioni dopo le prime applicazioni in campo medico descritte nel Capitolo 12. Una delle poche eccezioni è stato Jerry Feldman, che l'ha applicata a problemi nel campo della visione (Feldman e Yakimovsky, 1974) e della pianificazione (Feldman e Sproull, 1977). I sistemi esperti basati su regole della fine degli anni 1970 e dell'inizio degli anni 1980 si concentravano sul rispondere a domande, più che su prendere decisioni. I sistemi che raccomandavano azioni, generalmente lo facevano utilizzando regole condizione-azione anziché rappresentazioni esplicite di risultati e preferenze.

Le reti di decisione offrono un approccio molto più flessibile, per esempio consentendo alle preferenze di variare mentre il modello di transizione è mantenuto costante, o viceversa. Consentono anche un calcolo ben fondato di quali informazioni andare a cercare. Verso la fine degli anni 1980, grazie anche al lavoro di Pearl sulle reti bayesiane, i sistemi esperti basati sulla teoria delle decisioni furono ampiamente accettati (Horvitz *et al.*, 1988; Cowell *et al.*, 2002). In effetti, dal 1991 in poi la copertina della rivista *Artificial Intelligence* ha raffigurato una rete di decisione, anche se con qualche licenza artistica riguardo la direzione delle frecce.

I tentativi pratici di misurare l'utilità per gli esseri umani iniziarono con l'analisi delle decisioni nel periodo post-bellico (come discusso nei paragrafi precedenti). La misura di utilità del micromort è discussa in Howard (1989). Thaler (1992) rilevò che, considerando una probabilità di morte di 1/1000, una persona non sarebbe stata disposta a pagare più di 200 dollari per rimuovere il rischio, ma non avrebbe accettato 50.000 dollari per correre il rischio.

L'uso dei **QALY** (*quality-adjusted life year*) per svolgere analisi costi-benefici di interventi medici e politiche sociali correlate risale almeno al lavoro di Klarman *et al.* (1968), anche se il termine fu usato per

la prima volta da Zeckhauser e Shepard (1976). Come il denaro, i QALY corrispondono direttamente alle utilità soltanto sotto ipotesi piuttosto strette, come la neutralità al rischio, che sono spesso violate (Bere-sniak *et al.*, 2015); nondimeno, i QALY sono usati molto spesso nella pratica, per esempio per stabilire le politiche del servizio sanitario nazionale nel Regno Unito. Cfr. Russell (1990) per un tipico esempio di argomentazione in favore di un importante cambiamento nella politica di sanità pubblica sulla base di una maggiore utilità attesa misurata in QALY.

Keeney e Raiffa (1976) fornirono un'introduzione alla **teoria dell'utilità multiattributo**, descrivendo le prime implementazioni su computer dei metodi per elicitare i parametri necessari per una funzione di utilità multiattributo e fornendo ampi resoconti di applicazioni reali della teoria. Abbas (2018) esamina molti progressi compiuti a partire dal 1976. La teoria fu introdotta nell'IA principalmente con il lavoro di Wellman (1985), che studiò anche l'uso della dominanza stocastica e dei modelli di probabilità qualitativa (Wellman, 1988, 1990a). Wellman e Doyle (1992) offrirono uno studio preliminare di come un insieme complesso di relazioni di indipendenza fra le utilità possa essere usato per fornire un modello strutturato di una funzione di utilità, in modo molto simile a come le reti bayesiane forniscono un modello strutturato delle distribuzioni congiunte di probabilità. Bacchus e Grove (1995, 1996) e La Mura e Shoham (1999) presentarono ulteriori risultati su questa linea di ricerca. Boutilier *et al.* (2004) descrisse le reti CP-net, un formalismo grafico completo per esprimere preferenze condizionali *ceteris paribus*.

La **maledizione dell'ottimizzatore** fu portata all'attenzione degli analisti decisionali da Smith e Winkler (2006), che evidenziarono il fatto che i benefici finanziari per il cliente proiettati dagli analisti per le loro proposte non si materializzavano mai, e lo riconducessero direttamente al bias introdotto selezionando un'azione ottima e mostrando che un'analisi bayesiana più completa elimina il problema.

Lo stesso concetto di base è stato chiamato **disappunto post-decisione** da Harrison e March (1984) e fu notato nel contesto dell'analisi di progetti di investimento da parte di Brown (1974). La maledizione dell'ottimizzatore è anche strettamente correlata alla **maledizione del vincitore** (Capen *et al.*, 1971; Thaler, 1992), che vale per le offerte competitive nelle aste: chiunque vinca l'asta molto probabilmente ha sovrastimato il valore dell'oggetto acquistato. Capen *et al.* citano un ingegnere petrolifero sul tema delle aste dei

diritti di perforazione di pozzi di petrolio: “Se uno vince un'area contro due o tre altri può sentirsi fortunato, ma come dovrebbe sentirsi se vincesse contro 50 altri? Male”.

Il paradosso di Allais, dal nome dell'economista vincitore del Premio Nobel Maurice Allais (1953), fu verificato sperimentalmente per mostrare che le persone sono consistentemente inconsistenti nei loro giudizi (Tversky e Kahneman, 1982; Conlisk, 1989). Il paradosso di Ellsberg sull'avversione all'ambiguità fu introdotto nella tesi di dottorato di Daniel Ellsberg (1962).¹⁰ Fox e Tversky (1995) descrissero un altro studio sul tema. Machina (2005) fornì una panoramica sulla scelta in condizioni di incertezza e su come si distingua dalla teoria dell'utilità attesa. Il testo classico di Keeney e Raiffa (1976) e il lavoro più recente di Abbas (2018) offrono un'analisi approfondita delle preferenze in condizioni di incertezza.

Il 2009 fu un grande anno per i libri sull'**irrazionalità** umana, tra cui *Predictably Irrational* (Ariely, 2009), *Sway* (Brafman e Brafman, 2009), *Nudge* (Thaler e Sunstein, 2009), *Kluge* (Marcus, 2009), *How We Decide* (Lehrer, 2009) e *On Being Certain* (Burton, 2009), che fecero da complemento al classico *Judgment Under Uncertainty* (Kahneman *et al.*, 1982) e all'articolo che aveva iniziato gli studi sul tema (Kahneman e Tversky, 1979). Kahneman stesso fornisce un'interessante e godibile resoconto in *Pensieri lenti e veloci* (Kahneman, 2011).

Il campo della psicologia evoluzionista (Buss, 2005), d'altro canto, si è contrapposto a questa letteratura, sostenendo che gli esseri umani sono piuttosto razionali in contesti appropriati dal punto di vista evoluzionistico. I suoi sostenitori evidenziano che l'irrazionalità è penalizzata per definizione in un contesto evoluzionistico e mostrano che in alcuni casi è solo un artefatto delle condizioni sperimentali (Cummins e Allen, 1998). Recentemente c'è stata una ripresa di interesse nei modelli bayesiani della cognizione, dopo decenni di pessimismo (Elio, 2002; Chater e Oaksford, 2008; Griffiths *et al.*, 2008); ma non mancano i detrattori (Jones e Love, 2011).

La teoria del valore dell'informazione fu esaminata per la prima nel contesto di esperimenti statistici, dove fu usata una quasi-utilità (riduzione di entropia)

¹⁰ Ellsberg divenne in seguito un analista militare presso la RAND Corporation e lasciò filtrare i documenti noti come Pentagon Papers, contribuendo così alla fine della guerra del Vietnam.

(Lindley, 1956). Lo studioso di teoria del controllo Ruslan Stratonovich (1965) sviluppò la teoria più generale presentata qui, in cui l'informazione ha valore per la sua capacità di influire sulle decisioni. Il lavoro di Stratonovich rimase a lungo sconosciuto in Occidente, dove Ron Howard (1966) introdusse pionieristicamente la stessa idea. Il suo articolo termina con la frase: "Se la teoria del valore dell'informazione e le strutture di teoria delle decisioni a essa associate non occuperanno in futuro una gran parte della formazione degli ingegneri, vorrà dire che il ruolo tradizionale dell'ingegneria di gestire risorse scientifiche ed economiche per il beneficio dell'uomo sarà stato ceduto a un'altra professione". A tutt'oggi, una simile rivoluzione nei metodi manageriali non si è verificata.

L'algoritmo per la raccolta di informazioni miope descritto in questo capitolo è citato ovunque nella letteratura in tema di analisi delle decisioni; i suoi elementi di base si possono trovare nell'articolo originale sui diagrammi di influenza (Howard e Matheson, 1984). Metodi di calcolo efficiente sono stati studiati da Dittmer e Jensen (1997). Laskey (1995) e Nielsen e Jensen (2003) hanno discusso metodi per l'analisi di sensibilità in reti bayesiane e reti di decisione, rispettivamente. Il classico *Robust and Optimal Control* (Zhou *et al.*, 1995) fornì una trattazione completa e un confronto tra gli approcci robusti e di teoria delle decisioni al tema delle decisioni in condizioni di incertezza.

Il problema della caccia al tesoro fu risolto in modo indipendente da molti autori, risalendo almeno agli articoli sui test sequenziali di Gluss (1959) e Mitten (1960). Lo stile della dimostrazione fornita in questo capitolo si basa su un risultato di base dovuto a Smith (1956) che mette in relazione il valore di una sequenza al valore della stessa sequenza con due elementi adiacenti scambiati. Questi risultati per test indipendenti furono estesi ai problemi più generali di ricerca ad al-

bero e su grafi (in cui i test sono parzialmente ordinati) da Kadane e Simon (1977). Risultati sulla complessità di calcoli non miopi del valore dell'informazione sono stati ottenuti da Krause e Guestrin (2009). Krause *et al.* (2008) identificò dei casi in cui la submodularità conduce a un algoritmo di approssimazione trattabile, basandosi sul lavoro fondamentale di Nemhauser *et al.* (1978) sulle funzioni submodulari; Krause e Guestrin (2005) identificarono casi in cui un algoritmo di programmazione dinamica esatta fornisce una soluzione efficiente sia per la selezione di un sottoinsieme di evidenza sia per la generazione di un piano condizionale.

Harsanyi (1967) studiò il problema dell'informazione *incompleta* in teoria dei giochi, dove i giocatori non conoscono necessariamente con esattezza le funzioni di payoff degli avversari. Mostrò che tali giochi sono identici a giochi con informazione *imperfetta*, dove i giocatori sono incerti sullo stato del mondo, attraverso il trucco di aggiungere variabili di stato che fanno riferimento ai payoff dei giocatori. Cyert e de Groot (1979) svilupparono una teoria dell'**utilità adattativa** in cui un agente può essere incerto circa la propria funzione di utilità e può ottenere maggiori informazioni attraverso l'esperienza.

Il lavoro sull'elicitazione di preferenze bayesiana (Chajewska *et al.*, 2000; Boutilier, 2002) inizia dall'assumere di una probabilità a priori sulla funzione di utilità dell'agente. Fern *et al.* (2014) proposero un modello di teoria delle decisioni per l'**assistenza**, in cui un robot tenta di determinare e fornire assistenza su un obiettivo umano riguardo il quale è inizialmente incerto. L'esempio dello spegnimento descritto nel Paragrafo 16.7.2 è adattato da Hadfield-Menell *et al.* (2017b). Russell (2019) propone un quadro generale per un'IA benefica in cui il gioco del pulsante di spegnimento rappresenta un esempio fondamentale.

Decisioni complesse

- 17.1 Problemi di decisione sequenziali
- 17.2 Algoritmi per MDP
- 17.3 Problemi dei banditi
- 17.4 MDP parzialmente osservabili
- 17.5 Algoritmi per risolvere POMDP
- 17.6 Riepilogo
- Note storiche e bibliografiche

In cui esaminiamo alcuni metodi per decidere cosa fare oggi, posto che potremmo dover affrontare un'altra decisione domani.

In questo capitolo affrontiamo i problemi computazionali connessi al processo decisionale in un ambiente stocastico. Il Capitolo 16 si è concentrato su decisioni episodiche, in cui l'utilità dell'esito di ogni azione è ben nota: ora ci occupiamo dei **problemi di decisione sequenziali**, in cui l'utilità dell'agente dipende da una sequenza di decisioni. Questi problemi includono utilità, incertezza e percezione, e comprendono come caso speciale i problemi di ricerca e pianificazione. Il Paragrafo 17.1 spiega come sono definiti i problemi di decisione sequenziali, il Paragrafo 17.2 descrive alcuni metodi per risolverli producendo comportamenti appropriati per un ambiente stocastico. Il Paragrafo 17.3 tratta i **problemi dei banditi con molte braccia** (*multi-armed bandit*), una classe specifica e affascinante di problemi di decisione sequenziali che si presenta in molti contesti. Il Paragrafo 17.4 esamina i problemi di decisione in ambienti parzialmente osservabili e il Paragrafo 17.5 spiega come risolverli.

17.1 Problemi di decisione sequenziali

Supponiamo che un agente sia situato nell'ambiente 4×3 mostrato nella Figura 17.1(a). A partire dallo stato iniziale, l'agente deve scegliere un'azione in ogni passo temporale; l'interazione con l'ambiente termina quando l'agente raggiunge uno dei due stati obiettivo, marcati +1 e -1. Esattamente come per i problemi di ricerca, le azioni disponibili per l'agente in ogni stato sono date da $AZIONI(s)$, talvolta abbreviato in $A(s)$; nell'ambiente 4×3 , le azioni possibili in ogni stato sono *Su*, *Giù*, *Sinistra* e *Destra*. Per ora ipotizziamo che l'ambiente sia **completamente osservabile**, dimodoché l'agente saprà sempre dov'è.

Se l'ambiente fosse deterministico, la soluzione sarebbe facile: [*Su*, *Su*, *Destra*, *Destra*, *Destra*]. Sfortunatamente questa soluzione non è infallibile, perché le azioni sono inaffidabili.

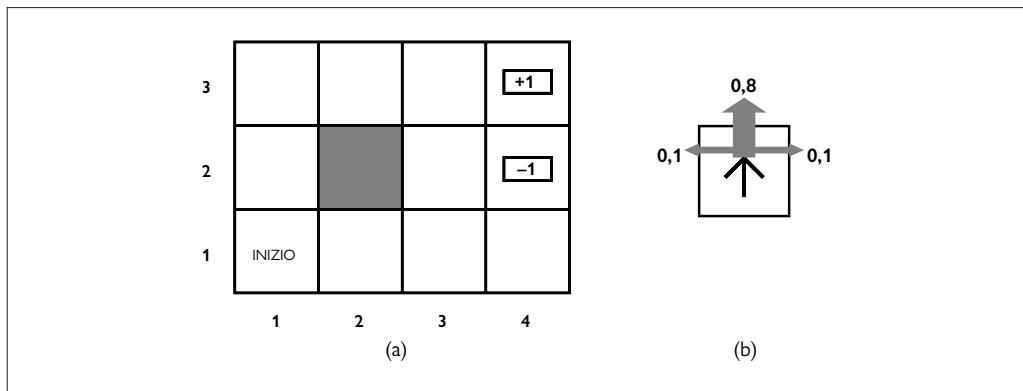


Figura 17.1 (a) un semplice ambiente stocastico 4×3 che presenta all’agente un problema di decisione sequenziale. (b) Il modello di transizione dell’ambiente: con probabilità 0,8 si ottiene l’esito desiderato, ma con probabilità 0,2 l’agente si muove in una direzione ortogonale a quella voluta. La collisione con un muro ha come risultato nessun movimento. Le transizioni nei due stati terminali hanno ricompensa +1 e -1, rispettivamente, mentre tutte le altre transizioni hanno ricompensa -0,04.

Il particolare modello di movimento stocastico che adottiamo è illustrato nella Figura 17.1(b). Ogni azione ha una probabilità 0,8 di ottenere l’effetto voluto, ma negli altri casi l’agente si muove ad angolo retto rispetto alla direzione prevista. Inoltre, se l’agente sbatte contro un muro rimane nella stessa posizione. Per esempio, dal riquadro di partenza (1,1), l’azione S_u sposta l’agente in (1,2) con probabilità 0,8, ma con probabilità 0,1 l’agente si sposta a destra in (2,1), e con probabilità 0,1 va a sinistra, sbatte contro il muro e rimane in (1,1). In un simile ambiente, la sequenza $[S_u, S_u, Destra, Destra, Destra]$ supera la barriera e raggiunge lo stato obiettivo in (4,3) con probabilità $0,8^5 = 0,32768$. C’è anche una piccola possibilità di raggiungere accidentalmente l’obiettivo passando all’altra parte: questo tragitto ha una probabilità $0,1^4 \times 0,8$, per una probabilità totale di 0,32776 (cfr. anche l’Esercizio 17.MDPX).

Come nel Capitolo 3, il **modello di transizione** (o semplicemente “modello”, quando il significato è chiaro) descrive l’esito di ogni azione in ogni stato. In questo caso l’esito è stocastico, perciò scriviamo $P(s'|s, a)$ per indicare la probabilità di raggiungere lo stato s' eseguendo l’azione a nello stato s (alcuni autori scrivono $T(s, a, s')$ per il modello di transizione). Presumeremo che le transizioni siano **markoviane**, ovvero che la probabilità di raggiungere s' partendo da s dipenda solo da s stesso e non dalla storia degli stati precedenti.

Per completare la definizione dell’ambiente operativo dobbiamo ancora specificare la funzione di utilità dell’agente. Dato che il problema è sequenziale, la funzione dipenderà da una sequenza di stati e azioni – una **storia dell’ambiente** – piuttosto che da uno stato singolo. Più avanti in questo paragrafo esamineremo la natura delle funzioni di utilità sulle storie; per ora ci limitiamo a dire che per ogni transizione da s a s' attraverso l’azione a , l’agente riceve una **ricompensa (reward)** $R(s, a, s')$. Le ricompense possono essere positive o negative, ma sono limitate da $\pm R_{\max}$ ¹.

Nel nostro particolare esempio la ricompensa è -0,04 per tutte le transizioni tranne quelle che entrano in stati terminali (che hanno ricompensa +1 e -1). L’utilità della storia dell’ambiente è pari (per adesso) alla somma delle ricompense ricevute; se per esempio l’agente raggiunge lo stato +1 dopo 10 passi, la sua utilità totale sarà $(9 \times -0,04) + 1 = 0,64$. La ricompensa negativa di -0,04 spinge l’agente a raggiungere velocemente (4,3), cosicché il nostro

ricompensa

¹ È anche possibile utilizzare i costi $c(s, a, s')$ come abbiamo fatto per la definizione dei problemi di ricerca nel Capitolo 3. L’uso delle ricompense, tuttavia, è diventato standard nella letteratura sulle decisioni sequenziali in condizioni di incertezza.

ambiente rappresenta una generalizzazione stocastica dei problemi di ricerca del Capitolo 3. Un altro modo di esprimere lo stesso concetto è dire che l'agente non gradisce di stare in questo ambiente e vuole uscire dal gioco il più in fretta possibile.

Per riassumere: un problema di decisione sequenziale per un ambiente stocastico completamente osservabile con un modello di transizione markoviano e ricompense additive prende il nome di **processo decisionale di Markov** o **MDP** (*Markov decision process*) ed è costituito da un insieme di stati (con uno stato iniziale s_0); un modello di transizione $P(s'|s, a)$; e una funzione ricompensa $R(s, a, s')$. I metodi per risolvere i problemi MDP solitamente coinvolgono la **programmazione dinamica**: semplificare un problema suddividendolo ricorsivamente in porzioni più piccole e ricordare le soluzioni ottime dei sottoproblemi.

processo
decisionale
di Markov

programmazione
dinamica

La domanda successiva è: che aspetto ha una soluzione a tale problema? Nessuna sequenza prefissata di azioni è in grado di risolvere il problema, perché l'agente potrebbe ritrovarsi in uno stato diverso dall'obiettivo. Ne consegue che la soluzione dovrà specificare quello che l'agente deve fare in *ogni* stato potenzialmente raggiungibile. Una soluzione di questo tipo si chiama **politica**. È consuetudine indicare la politica con la lettera π , mentre $\pi(s)$ è l'azione raccomandata dalla politica π nello stato s . Indipendentemente dall'esito dell'azione, lo stato risultante sarà incluso nella politica e l'agente saprà che cosa fare in seguito.

politica

Ogni volta che una data politica viene eseguita a partire dallo stato iniziale, la natura stocastica dell'ambiente potrebbe portare a una storia diversa. La qualità della politica va quindi misurata in base all'utilità *attesa* delle possibili storie dell'ambiente da essa generate. Una **politica ottima**, indicata con π^* , è una politica che fornisce la più alta utilità attesa. Data π^* , l'agente decide cosa fare guardando la sua percezione corrente, che gli dice qual è lo stato corrente s , e quindi esegue l'azione $\pi^*(s)$. Una politica rappresenta la funzione agente in modo esplicito ed è quindi la descrizione di un agente reattivo semplice, calcolata a partire dall'informazione utilizzata da un agente basato sull'utilità.

politica ottima

Le politiche ottime per il mondo della Figura 17.1 sono mostrate nella Figura 17.2(a). Ci sono due politiche perché l'agente è perfettamente indifferente tra andare a sinistra e in alto quando si trova in (3,1): andare a sinistra è più sicuro ma più lungo, mentre andare in alto è più veloce ma presenta il rischio di cadere in (4,2) per sbaglio. In generale ci saranno molteplici politiche ottime.

L'equilibrio tra rischio e ricompensa cambia in base al valore di $r = R(s, a, s')$ per transizioni tra stati non terminali. Le politiche mostrate nella Figura 17.2(a) sono ottime per

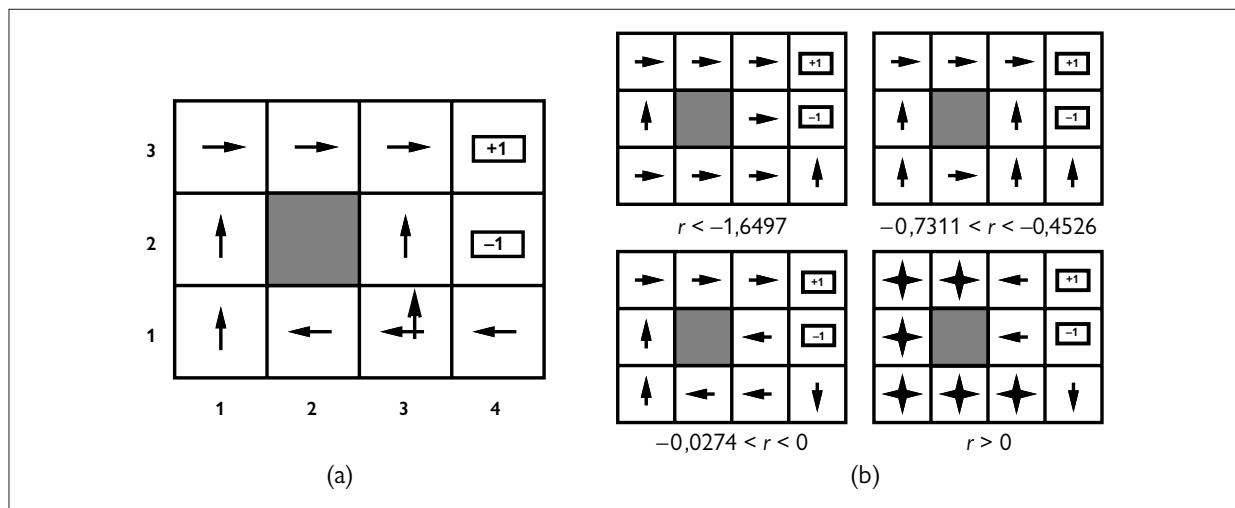


Figura 17.2 (a) Le politiche ottime per l'ambiente stocastico con $r = -0,04$ per transizioni tra stati non terminali. Ci sono due politiche perché nello stato (3,1) Sinistra e Su sono entrambe ottime. (b) Politiche ottime per quattro diversi intervalli di r .

$-0,0850 < r < -0,0273$. La Figura 17.2(b) riporta le politiche ottime per altri intervalli di $r(s)$. Quando $r < -1,6497$, la vita è così dolorosa che l’agente punta dritto verso l’uscita più vicina, anche quando vale -1 . Quando $-0,7311 < r < -0,4526$, la vita è parecchio sgradevole; l’agente sceglie la via più diretta per arrivare allo stato $+1$ da $(2,1), (3,1)$ e $(3,2)$, ma da $(4,1)$ il costo per raggiungere $+1$ è talmente alto che preferisce cadere direttamente in -1 . Quando la vita è solo appena fastidiosa ($-0,0274 < r < 0$), la politica ottima non è disposta a correre *alcun rischio*: in $(4,1)$ e $(3,2)$ l’agente punta nella direzione direttamente opposta all’uscita di valore -1 in modo da non poterci cadere dentro per sbaglio, anche se questo significa sbattere la testa contro il muro parecchie volte. Infine, se $r > 0$, la vita diventa piacevole e l’agente evita *entrambe* le uscite: finché le azioni in $(4,1), (3,2)$ e $(3,3)$ sono quelle indicate, ogni politica è ottima, e l’agente ottiene una ricompensa totale infinita semplicemente non entrando mai in uno stato terminale. Risulta che ci sono nove politiche ottime per tutti gli intervalli di valori di r ; l’Esercizio 17.THRC vi chiederà di trovarle tutte.

L’introduzione dell’incertezza porta gli MDP più vicino al mondo reale rispetto ai problemi di ricerca deterministici. Per questa ragione gli MDP sono stati studiati in molti campi, dall’IA alla ricerca operativa, dall’economia alla teoria del controllo. Sono state proposte dozzine di algoritmi di soluzione, alcuni dei quali li esamineremo nel Paragrafo 17.2. Prima, però, esaminiamo in maggiore dettaglio le definizioni di utilità, politiche ottime e modelli per MDP.

17.1.1 Utilità nel tempo

Nell’esempio di MDP della Figura 17.1, la prestazione dell’agente è misurata sommando le ricompense per le transazioni effettuate: questa non è una scelta arbitraria, ma non è l’unica possibilità per la funzione di utilità² sulle storie dell’ambiente, che scriviamo $U_h([s_0, a_0, s_1, a_1, \dots, s_n])$.

La prima questione da considerare è se il processo di decisione preveda un **orizzonte finito o infinito**. Nel primo caso c’è un tempo *prefissato N* dopo il quale nulla importa più: il gioco, per così dire, è finito. Quindi:

$$U_h([s_0, a_0, s_1, a_1, \dots, s_{N+k}]) = U_h([s_0, a_0, s_1, a_1, \dots, s_N])$$

per ogni $k > 0$. Per esempio, ammettiamo che un agente parta nella posizione $(3,1)$ nel mondo 4×3 della Figura 17.1, e supponiamo $N = 3$. In questo caso, per avere una qualche possibilità di raggiungere lo stato $+1$ l’agente deve puntare direttamente verso di esso, e quindi l’azione ottima sarà andare *Su*. D’altro canto, con $N = 100$ ci sarà tutto il tempo di prendere il tragitto sicuro andando a *Sinistra*. Così, con un orizzonte finito, un’azione ottima in un dato stato potrebbe dipendere dal tempo rimasto. Una politica che dipende dal tempo è detta **non stazionaria**.

Quando non c’è un limite temporale prefissato, d’altra parte, non c’è alcuna ragione di comportarsi diversamente nello stesso stato in istanti differenti. Un’azione ottima quindi dipende solo dallo stato corrente, e la politica ottima è **stazionaria**. Nel caso a orizzonte infinito le politiche sono quindi più semplici, e in questo capitolo ci occuperemo più che altro di queste (vedremo più avanti che, per ambienti parzialmente osservabili, il caso a orizzonte infinito non è così semplice). Notate che “orizzonte infinito” non significa necessariamente che tutte le sequenze di stati siano infinite, ma solo che non c’è un limite prefissato. In un MDP a orizzonte infinito possono esserci sequenze finite di stati che contengono uno stato terminale.

La questione successiva da affrontare è il calcolo dell’utilità delle sequenze di stati. In questo capitolo utilizzeremo **ricompense scontate additive**: l’utilità di una storia è:

$$U_h([s_0, a_0, s_1, a_1, s_2, \dots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \dots,$$

orizzonte finito
orizzonte infinito


politica non
stazionaria
politica stazionaria

ricompensa
scontata additiva

² In questo capitolo utilizzeremo U per indicare la funzione di utilità (per coerenza con il resto del libro), ma in molti lavori sugli MDP si utilizza invece V (per *valore*).

dove il **fattore di sconto** γ è un numero compreso tra 0 e 1. Il fattore di sconto rappresenta la preferenza dell'agente per una ricompensa immediata rispetto a quelle future. Quando γ è vicino a 0, le ricompense in un futuro distante sono considerate insignificanti. Quando γ è vicino a 1, un agente è maggiormente disponibile ad attendere ricompense nel lungo termine. Quando γ è esattamente uguale a 1, le ricompense scontate si riducono al caso speciale delle **ricompense puramente additive**. Notate che l'additività è già stata usata implicitamente nelle funzioni di costo di cammino utilizzate negli algoritmi di ricerca euristici (Capitolo 3).

L'uso di ricompense scontate additive è sensato per diverse ragioni. Una è empirica: gli esseri umani e anche gli animali tendono ad attribuire maggior valore alle ricompense nel breve termine rispetto a quelle nel lungo termine. Un'altra è economica: se le ricompense sono in denaro, è effettivamente meglio ottenerle il prima possibile, perché così è possibile investirle e generare rendimenti mentre si attendono altre ricompense in futuro. In tale contesto, un fattore di sconto γ è equivalente a un tasso di interesse $(1/\gamma) - 1$. Per esempio, un fattore di sconto $\gamma = 0,9$ è equivalente a un tasso di interesse dell'11,1%.

Una terza ragione è l'incertezza sulle vere ricompense, che potrebbero anche non realizzarsi mai per vari motivi non considerati nel modello di transizione. Sotto certe ipotesi, un fattore di sconto γ è equivalente ad aggiungere una probabilità $1 - \gamma$ di una terminazione accidentale a ogni passo temporale, indipendentemente dall'azione effettuata.

Una quarta giustificazione deriva da una proprietà naturale delle preferenze sulle storie. Nella terminologia della teoria dell'utilità multiattributo (cfr. Paragrafo 16.4), ogni transizione $s_t \xrightarrow{a_t} s_{t+1}$ può essere vista come un **attributo** della storia $[s_0, a_0, s_1, a_1, s_2, \dots]$. In linea di principio, la funzione di utilità potrebbe dipendere da questi attributi in modi di complessità arbitraria, tuttavia esiste un'ipotesi altamente plausibile di indipendenza delle preferenze, per cui le preferenze dell'agente sulle sequenze di stati sono **stazionarie**.

Supponiamo che due storie $[s_0, a_0, s_1, a_1, s_2, \dots]$ e $[s'_0, a'_0, s'_1, a'_1, s'_2, \dots]$ inizino con la stessa transizione (cioè $s_0 = s'_0$, $a_0 = a'_0$, e $s_1 = s'_1$). Allora, per la stazionarietà delle preferenze, le due storie dovrebbero essere ordinate secondo le preferenze nello stesso modo delle storie $[s_1, a_1, s_2, \dots]$ e $[s'_1, a'_1, s'_2, \dots]$. Per dirla in termini più semplici, questo significa che, se preferite un futuro a un altro che comincerà da domani, dovreste continuare a preferire quel futuro anche se cominciasse da oggi. La stazionarietà è un'ipotesi che sembra innocua, ma lo sconto additivo è l'unica forma di utilità sulle storie che la soddisfa.

Un'ultima ragione che giustifica l'uso delle ricompense scontate è che consente di far scomparire alcuni problemi di infinito. In effetti, nel caso di orizzonti infiniti c'è una potenziale difficoltà: se l'ambiente non contiene uno stato terminale, o se l'agente non ne raggiunge mai uno, tutte le storie risulteranno infinitamente lunghe e le utilità con ricompense additive non scontate saranno generalmente infinite. Mentre è facile concordare sul fatto che $+\infty$ è meglio di $-\infty$, confrontare due sequenze di stati con utilità $+\infty$ è più difficile. Esistono tre soluzioni, delle quali ne abbiamo già considerate due.

- Utilizzando ricompense scontate, l'utilità di una sequenza infinita risulta *finita*. In effetti, se le ricompense sono limitate da $\pm R_{\max}$, abbiamo:

$$U_h([s_0, a_0, s_1, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1 - \gamma}, \quad (17.1)$$

usando la formula standard per la somma di una serie geometrica infinita.

- Se l'ambiente contiene stati terminali *ed è garantito che l'agente prima o poi ne raggiunga uno*, non dovremo mai confrontare sequenze infinite. Una politica che garantisce il raggiungimento di uno stato terminale si chiama **politica propria**. Se tutte le politiche sono proprie, possiamo usare $\gamma = 1$ (cioè, ricompense additive non scontate). Le prime tre politiche della Figura 17.2(b) sono proprie, ma la quarta non lo è: ottiene una ricompensa totale infinita rimanendo sempre fuori dagli stati terminali quando la ricompensa per le

fattore di sconto

**ricompensa
puramente additiva**

**preferenza
stazionaria**

politica propria

transizioni tra stati non terminali è positiva. L'esistenza di politiche improprie può causare il fallimento degli algoritmi standard per la risoluzione degli MDP quando le ricompense sono additive, e questo costituisce una buona ragione per utilizzare ricompense scontate.

ricompensa media

3. È possibile confrontare sequenze infinite in base alla **ricompensa media** ottenuta per ogni passo temporale. Supponiamo che le transizioni verso la casella (1,1) nel mondo 4×3 abbiano una ricompensa di 0,1 mentre le transizioni verso altri stati non terminali abbiano una ricompensa di 0,01. In questo caso una politica che fa del suo meglio per rimanere in (1,1) avrà una ricompensa media più alta di una che se ne va altrove. La ricompensa media è un criterio utile in alcuni problemi, ma l'analisi dei relativi algoritmi è un tema complesso.

L'approccio che utilizza le ricompense scontate additive è quello che presenta meno difficoltà nella valutazione delle storie, perciò lo utilizzeremo da qui in avanti.

17.1.2 Politiche ottime e utilità degli stati

Avendo deciso che l'utilità di una data storia è la somma delle ricompense scontate, possiamo confrontare le politiche mediante le utilità *attese* ottenute con la loro esecuzione. Ipotizziamo che l'agente si trovi in uno stato iniziale s e definiamo con S_t (una variabile casuale) lo stato che l'agente raggiunge al tempo t quando esegue una particolare politica π (naturalmente $S_0 = s$, lo stato in cui l'agente si trova ora). La distribuzione di probabilità sulle sequenze di stati S_1, S_2, \dots , è determinata dallo stato iniziale s , dalla politica π e dal modello di transizione per l'ambiente.

L'utilità attesa ottenuta con l'esecuzione della politica π a partire da s è data da:

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right], \quad (17.2)$$

dove l'aspettativa E è riferita alla distribuzione di probabilità sulle sequenze di stati determinate da s e π . Ora, di tutte le politiche che l'agente potrebbe scegliere di eseguire a partire da s , una o più avranno utilità attesa superiore a tutte le altre. Indicheremo una di queste politiche con π_s^* :

$$\pi_s^* = \operatorname{argmax}_{\pi} U^\pi(s). \quad (17.3)$$

Ricordiamo che π_s^* è una politica, perciò raccomanda un'azione per ogni stato; il suo legame con s in particolare è che è un politica ottima quando s è lo stato di partenza. Un'importante conseguenza dell'uso di utilità scontate con orizzonti infiniti è che la politica ottima è *indipendente* dallo stato iniziale (naturalmente la *sequenza di azioni* non sarà indipendente; ricordiamo che una politica è una funzione che specifica un'azione per ogni stato). Questo fatto appare intuitivamente ovvio: se la politica π_a^* è ottima a partire da a e la politica π_b^* , è ottima a partire da b , allora, quando queste politiche raggiungono un terzo stato c , non vi è motivo per cui non concordino tra loro, o con la politica π_c^* , su che cosa fare da lì in poi.³ Possiamo quindi indicare semplicemente con π^* una politica ottima.

Data questa definizione la vera utilità di uno stato non è altro che $U^{\pi^*}(s)$, ovvero la somma attesa delle ricompense scontate ottenute se l'agente esegue una politica ottima. La indichiamo con $U(s)$ per coerenza con la notazione usata nel Capitolo 16 per l'utilità di un esito. La Figura 17.3 mostra le utilità per il mondo 4×3 . Notate che i valori crescono per gli stati più vicini all'uscita +1, perché bastano meno passi per raggiungerla.

³ Benché questo sembri ovvio, non vale per politiche a orizzonte finito o per altri modi di combinare le ricompense nel tempo, come quello che considera il valore massimo. La dimostrazione segue direttamente dall'unicità della funzione di utilità sugli stati, come è mostrato nel Paragrafo 17.2.1.

	3	0,8516	0,9078	0,9578	+1
	2	0,8016		0,7003	-1
	1	0,7453	0,6953	0,6514	0,4279
	1	2	3	4	

Figura 17.3
Le utilità degli stati nel mondo 4×3 , con $\gamma = 1$ e $r = -0,04$ per transizioni verso stati non terminali.

La funzione di utilità $U(s)$ permette all'agente di scegliere le azioni in base al principio della massima utilità attesa descritto nel Capitolo 16, cioè scegliendo l'azione che massimizza la ricompensa per il passo successivo più l'utilità scontata attesa dello stato successivo:

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s')]. \quad (17.4)$$

Abbiamo definito l'utilità di uno stato come la somma attesa delle ricompense scontate da quel punto in avanti, quindi dev'esserci una relazione diretta con l'utilità degli stati vicini: *l'utilità di uno stato è la ricompensa attesa per la successiva transizione sommata all'utilità scontata dello stato successivo, assumendo che l'agente scelga l'azione ottima*. Questo significa che l'utilità è data da:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s')]. \quad (17.5)$$

Questa è l'**equazione di Bellman**, chiamata così in onore di Richard Bellman (1957). Le utilità degli stati, definite dall'Equazione (17.2) come le utilità attese delle sequenze di stati a essi successivi, corrispondono a soluzioni dell'insieme delle equazioni di Bellman. In effetti, esse sono le *uniche* soluzioni, come vedremo nel Paragrafo 17.2.1.



equazione
di Bellman

Esaminiamo una delle equazioni di Bellman per il mondo 4×3 . L'espressione per lo stato (1,1) è:

$$\begin{aligned} \max\{ & [0,8(-0,04 + \gamma U(1, 2)) + 0,1(-0,04 + \gamma U(2, 1)) + 0,1(-0,04 + \gamma U(1, 1))], \\ & [0,9(-0,04 + \gamma U(1, 1)) + 0,1(-0,04 + \gamma U(1, 2))], \\ & [0,9(-0,04 + \gamma U(1, 1)) + 0,1(-0,04 + \gamma U(2, 1))], \\ & [0,8(-0,04 + \gamma U(2, 1)) + 0,1(-0,04 + \gamma U(1, 2)) + 0,1(-0,04 + \gamma U(1, 1))] \} \end{aligned}$$

dove le quattro espressioni interne corrispondono a *Su*, *Sinistra*, *Giù* e *Destra*. Inserendo i numeri della Figura 17.3, con $\gamma = 1$, troviamo che *Su* è la migliore azione.

Un'altra quantità importante è la **funzione azione-utilità**, o **funzione-Q**: $Q(s, a)$ è l'utilità attesa di effettuare una data azione in un dato stato. La funzione-Q è legata alle utilità in modo ovvio:

$$U(s) = \max_a Q(s, a). \quad (17.6)$$

Inoltre, la politica ottima può essere ricavata dalla funzione-Q nel modo seguente:

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a). \quad (17.7)$$

funzione-Q

Possiamo anche sviluppare un’equazione di Bellman per funzioni-Q, osservando che la ricompensa totale attesa per l’esecuzione di un’azione è data dalla sua ricompensa immediata sommata all’utilità scontata dello stato risultato, che a sua volta può essere espressa in termini della funzione-Q:

$$\begin{aligned} Q(s, a) &= \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma U(s')] \\ &= \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma \max_{a'} Q(s', a')] \end{aligned} \quad (17.8)$$

Risolvendo le equazioni di Bellman per U (o per Q) otteniamo ciò che serve per trovare una politica ottima. La funzione-Q compare molto spesso negli algoritmi per risolvere problemi MDP, perciò utilizzeremo la definizione seguente:

```
function VALORE-Q(mdp, s, a, U) returns un valore di utilità
return  $\sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma U[s']]$ 
```

17.1.3 Scale di ricompensa

Nel Capitolo 16 abbiamo visto che la scala delle utilità è arbitraria: una trasformazione affine lascia la decisione ottima invariata. Possiamo sostituire $U(s)$ con $U'(s) = mU(s) + b$ dove m e b sono costanti qualsiasi tali che $m > 0$. È facile vedere, dalla definizione di utilità come somma scontata di ricompense, che una trasformazione simile applicata alle ricompense lascerà invariata la politica ottima in un MDP:

$$R'(s, a, s') = mR(s, a, s') + b.$$

**teorema
dello shaping**

Tuttavia, la scomposizione delle utilità in ricompense additive lascia molta più libertà nella definizione delle ricompense. Sia $\Phi(s)$ una *qualsiasi* funzione dello stato s . Allora, secondo il **teorema dello shaping** (o del *modellamento*), la trasformazione seguente lascia invariata la politica ottima:

$$R'(s, a, s') = R(s, a, s') + \gamma\Phi(s') - \Phi(s). \quad (17.9)$$

Per dimostrarlo, dobbiamo dimostrare che due MDP, M e M' , hanno politiche ottime identiche purché differiscano soltanto per le loro funzioni di ricompensa come specificato nell’Equazione (17.9). Iniziamo dall’equazione di Bellman per Q , la funzione-Q per l’MDP M :

$$Q(s, a) = \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma \max_{a'} Q(s', a')].$$

Ora, poniamo $Q'(s, a) = Q(s, a) - \Phi(s)$ e inseriamolo nella precedente equazione, ottenendo:

$$Q'(s, a) + \Phi(s) = \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma \max_{a'} (Q(s', a') + \Phi(s'))].$$

che si semplifica in:

$$\begin{aligned} Q'(s, a) &= \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma\Phi(s') - \Phi(s) + \gamma \max_{a'} Q'(s', a')] \\ &= \sum_{s'} P(s' | s, a)[R'(s, a, s') + \gamma \max_{a'} Q'(s', a')]. \end{aligned}$$

In altre parole, $Q'(s, a)$ soddisfa l’equazione di Bellman per l’MDP M' . Ora possiamo ricavare la politica ottima per M' usando l’Equazione (17.7):

$$\pi_M^*(s) = \operatorname{argmax}_a Q'(s, a) = \operatorname{argmax}_a Q(s, a) - \Phi(s) = \operatorname{argmax}_a Q(s, a) = \pi_M^*(s).$$

La funzione $\Phi(s)$ è spesso chiamata **potenziale**, per analogia con il potenziale elettrico (voltage) che genera i campi elettrici. Il termine $\gamma\Phi(s') - \Phi(s)$ fa da gradiente del potenziale.

Quindi, se $\Phi(s)$ ha valore più elevato in stati con utilità più alta, l'aggiunta di $\gamma\Phi(s') - \Phi(s)$ alla ricompensa ha l'effetto di condurre l'agente "in salita" nell'utilità.

A prima vista potrebbe sembrare contorto il fatto di poter modificare in questo modo la ricompensa senza cambiare la politica ottima. Per capire meglio è utile ricordare che *tutte le politiche sono ottime* con una funzione di ricompensa che vale zero ovunque. Questo significa che, secondo il teorema dello shaping, tutte le politiche sono ottime per qualsiasi ricompensa basata su potenziale della forma $R(s, a, s') = \gamma\Phi(s') - \Phi(s)$. Intuitivamente, questo vale perché con una tale ricompensa non importa in quale modo l'agente vada da A a B (è più facile vederlo quando $\gamma = 1$: lungo qualsiasi cammino la somma delle ricompense si riduce a $\Phi(B) - \Phi(A)$, perciò tutti i cammini sono equivalenti). Perciò, aggiungendo una ricompensa basata su potenziale a qualsiasi altra ricompensa non si dovrebbe modificare la politica ottima.

Grazie alla flessibilità offerta dal teorema dello shaping possiamo in effetti dare una mano all'agente facendo in modo che la ricompensa immediata rifletta più direttamente ciò che l'agente deve fare. In effetti, se poniamo $\Phi(s)=U(s)$, allora la politica greedy π_G rispetto alla ricompensa modificata R' è anche una politica ottima:

$$\begin{aligned}\pi_G(s) &= \operatorname{argmax}_a \sum_{s'} P(s' | s, a) R'(s, a, s') \\ &= \operatorname{argmax}_a \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma\Phi(s') - \Phi(s)] \\ &= \operatorname{argmax}_a \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s') - U(s)] \\ &= \operatorname{argmax}_a \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s')] \\ &= \pi^*(s) \quad (\text{per l'Equazione (17.4)}).\end{aligned}$$

Naturalmente, per fissare $\Phi(s)=U(s)$ dovremmo conoscere $U(s)$; perciò non ci sono pasti gratis, ma rimane comunque importante definire una funzione di ricompensa che sia d'aiuto per quanto possibile. È così che fanno gli addestratori di animali quando presentano al soggetto una piccola ricompensa a ogni passo della sequenza obiettivo.

17.1.4 Rappresentazione di MDP

Il modo più semplice per rappresentare $P(s' | s, a)$ e $R(s, a, s')$ è quello di utilizzare grandi tabelle tridimensionali di dimensione $|S|^2|A|$. Funziona per problemi piccoli come il mondo 4×3 , per cui le tabelle hanno $11^2 \times 4 = 484$ elementi ciascuna. In alcuni casi le tabelle sono **sparse** – nel senso che la maggior parte degli elementi sono zero perché da ogni stato s si può passare soltanto in un numero limitato di stati s' – il che significa che le tabelle hanno dimensione $O(|S||A|)$. Per problemi più grandi, tuttavia, anche le tabelle sparse diventano troppo grandi.

Proprio come nel Capitolo 16, in cui abbiamo esteso le reti bayesiane con nodi di azione e di utilità per creare reti di decisione, possiamo rappresentare gli MDP con l'estensione di reti bayesiane dinamiche (DBN, cfr. Capitolo 14) con nodi di decisione, ricompensa e utilità per creare **reti di decisione dinamiche**, o DDN, che sono **rappresentazioni fattorizzate** nella terminologia del Capitolo 2; e che generalmente hanno un vantaggio di complessità esponenziale rispetto alle rappresentazioni atomiche e possono modellare problemi del mondo reale piuttosto sostanziosi.

rete di decisione
dinamica

La Figura 17.4, basata sulla DBN della Figura 14.13(b), mostra alcuni elementi di un modello leggermente realistico di un robot mobile che è in grado di ricaricarsi. Lo stato S_t è composto di quattro variabili di stato:

- \mathbf{X}_t consiste nella posizione bidimensionale su una griglia più l'orientamento;
- $\dot{\mathbf{X}}_t$ è il tasso di variazione di \mathbf{X}_t ;

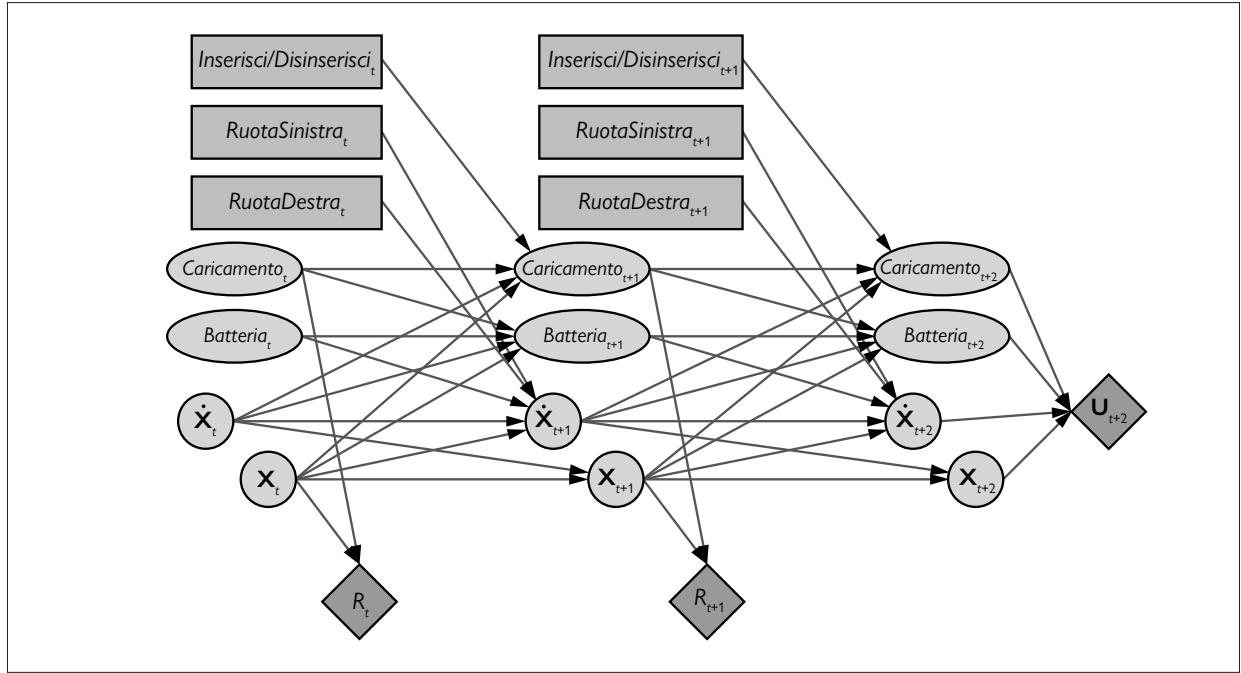


Figura 17.4 Una rete di decisione dinamica per un robot mobile con variabili di stato per livello della batteria, stato di ricarica in corso, posizione e velocità, e variabili di azione per i motori delle ruote di sinistra e destra e per la ricarica.

- $Caricamento_t$ è vero quando il robot è collegato a una sorgente di energia elettrica;
- $Batteria_t$ è il livello di carica della batteria, che modelliamo come numero intero nell'intervallo $0, \dots, 5$.

Lo spazio degli stati per l'MDP è il prodotto cartesiano degli intervalli di queste quattro variabili. L'azione ora è un insieme \mathbf{A}_t di variabili d'azione i cui elementi sono *Inserisci/Disinserisci*, che ha tre valori (*inserito*, *disinserito* e *noop*); *RuotaSinistra* per la potenza fornita alla ruota sinistra e *RuotaDestra* per la potenza fornita alla ruota destra. L'insieme delle azioni per l'MDP è il prodotto cartesiano degli intervalli di queste tre variabili. Notate che ogni variabile d'azione ha effetto soltanto su un sottoinsieme delle variabili di stato.

Il modello di transizione complessivo è la distribuzione condizionale $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, \mathbf{A}_t)$, che può essere calcolata come prodotto di probabilità condizionali dalla DDN. La ricompensa in questo caso è una singola variabile che dipende soltanto dalla posizione \mathbf{X} (si ottiene per l'arrivo a destinazione) e da *Caricamento*, dato che il robot deve pagare per l'uso dell'energia elettrica. In questo particolare modello, la ricompensa non dipende dall'azione o dallo stato risultato.

La rete della Figura 17.4 è stata proiettata due passi in avanti nel futuro. Notate che include nodi per le *ricompense* per t e $t + 1$, ma l'*utilità* per $t + 2$. Il motivo è che l'agente deve massimizzare la somma (scontata) di tutte le ricompense future, e $U(\mathbf{X}_{t+2})$ rappresenta tutte le ricompense da $t + 2$ in avanti. Se è disponibile un'approssimazione euristica per U , può essere inclusa nella rappresentazione dell'MDP in questo modo e usata al posto di un'ulteriore espansione. Questo approccio è strettamente legato all'uso della ricerca in profondità limitata e delle funzioni di valutazione euristiche per i giochi che abbiamo visto nel Capitolo 5.

Un altro MDP interessante e ben studiato è il gioco del Tetris (Figura 17.5(a)). Le variabili di stato sono *PezzoCorrente*, *PezzoSuccessivo*, e una variabile il cui valore è costituito

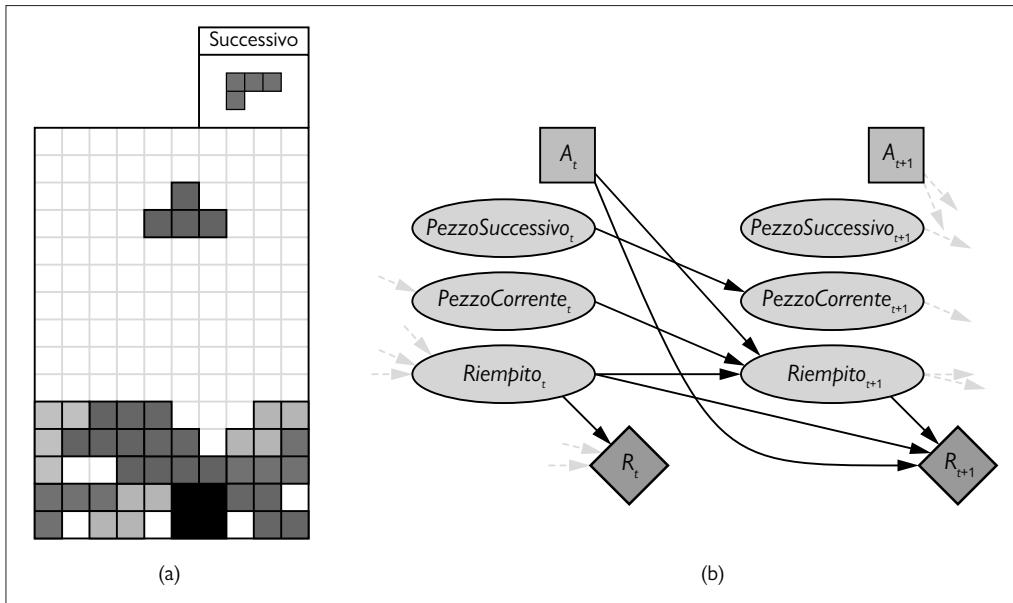


Figura 17.5 (a) Il gioco del Tetris. Il pezzo con forma a T in alto al centro può essere lasciato cadere con qualsiasi orientamento e in qualsiasi posizione orizzontale. Se una riga è completata scompare, le righe sopra di essa scorrono di un posto verso il basso, e l'agente riceve un punto. Il pezzo successivo (in questo caso è il pezzo a forma di L in alto a destra) diventa quello corrente e ne appare uno nuovo, scelto a caso dai sette tipi di pezzi disponibili. Il gioco termina se la griglia si riempie fino in alto. (b) La DDN per l'MDP del Tetris.

da un vettore di bit denominata *Riempito* con un bit per ognuna delle 10×20 posizioni nella griglia. Lo spazio degli stati ha dunque $7 \times 7 \times 2^{200} \approx 10^{62}$ stati. La DDN per il Tetris è mostrata nella Figura 17.5(b). Notate che $Riempito_{t+1}$ è una funzione deterministica di $Riempito_t$ e A_t . Risulta che ogni politica per Tetris è propria (raggiunge uno stato terminale): alla fine la griglia si riempie nonostante i migliori sforzi profusi per svuotarla.

17.2 Algoritmi per MDP

In questo paragrafo presentiamo quattro diversi algoritmi per la risoluzione di problemi MDP. I primi tre, **iterazione dei valori**, **iterazione delle politiche** e **programmazione lineare**, generano soluzioni esatte offline. Il quarto è in realtà una famiglia di algoritmi approssimati online che includono la **pianificazione Monte Carlo**.

pianificazione
Monte Carlo

17.2.1 Iterazione dei valori

L'equazione di Bellman (Equazione (17.5)) è alla base dell'algoritmo di **iterazione dei valori** per la risoluzione di MDP. Se ci sono n possibili stati, ci devono essere n equazioni di Bellman, una per ogni stato. Le n equazioni contengono n incognite: le utilità degli stati. Per trovare le utilità quindi vorremmo risolvere le equazioni contemporaneamente, ma c'è un problema: le equazioni sono *non lineari*, perché l'operatore “max” non è lineare. Mentre i sistemi di equazioni lineari possono essere risolti facilmente con tecniche algebriche, quelli non lineari sono più problematici. Un possibile approccio è quello *iterativo*: cominciamo con dei valori iniziali arbitrari delle utilità, calcoliamo la parte destra dell'equazione e la sostituiamo nella parte sinistra, aggiornando così l'utilità di ogni stato in base a quelle dei suoi vicini. Il processo viene ripetuto finché non si raggiunge un equilibrio.

iterazione dei valori

```

function ITERAZIONE-VALORI(mdp,  $\varepsilon$ ) returns una funzione di utilità
  inputs: mdp, un MDP con stati  $S$ , azioni  $A(s)$ , modello di transizione  $P(s'|s,a)$ ,
    ricompense  $R(s,a, s')$  e sconto  $\gamma$ 
     $\varepsilon$ , l'errore massimo consentito nell'utilità di ogni stato
  local variables:  $U, U'$ , vettori di utilità per gli stati in  $S$ , inizialmente a zero
     $\delta$ , la massima variazione relativa di utilità per qualsiasi stato a ogni iterazione
  repeat
     $U \leftarrow U'$ ;  $\delta \leftarrow 0$ 
    for each stato  $s$  in  $S$  do
       $U'[s] \leftarrow \max_{a \in A(s)} \text{VALORE-Q}(\text{mdp}, s, a, U)$ 
      if  $|U'[s] - U[s]| > \delta$  then  $\delta \leftarrow |U'[s] - U[s]|$ 
    until  $\delta \leq \varepsilon (1 - \gamma)/\gamma$ 
    return  $U$ 

```

Figura 17.6 L'algoritmo di iterazione dei valori per il calcolo delle utilità degli stati. La condizione di terminazione è presa dall'Equazione (17.12).

aggiornamento
di Bellman

Sia $U_i(s)$ il valore di utilità dello stato s alla i -esima iterazione. Il passo di iterazione, chiamato **aggiornamento di Bellman**, ha quest'aspetto:

$$U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U_i(s')], \quad (17.10)$$

dove si ipotizza che l'aggiornamento sia applicato simultaneamente a tutti gli stati in ogni iterazione. Se applichiamo l'aggiornamento di Bellman un numero infinito di volte, raggiungeremo sicuramente un equilibrio (cfr. il prossimo sottoparagrafo dedicato alla convergenza dell'iterazione dei valori), nel qual caso i valori finali di utilità dovranno essere le soluzioni delle equazioni di Bellman. In effetti essi sono anche le *uniche* soluzioni, e la politica corrispondente (ottenuta per mezzo dell'Equazione (17.4)) è quella ottima. L'algoritmo dettagliato, che include una condizione di terminazione quando le utilità sono “sufficientemente vicine”, è riportato nella Figura 17.6. Notate l'utilizzo della funzione VALORE-Q definita in precedenza nel Paragrafo 17.1.2.

Possiamo applicare l'iterazione dei valori al mondo 4×3 della Figura 17.1(a). Cominciando con valori iniziali pari a zero, le utilità evolvono come mostra la Figura 17.7(a). Notate come gli stati che si trovano a diverse distanze da (4,3) accumulano ricompense negative finché non si trova un cammino per (4,3), dopodiché cominciano ad aumentare. Possiamo pensare che l'algoritmo di iterazione dei valori *propaghi informazione* attraverso lo spazio degli stati per mezzo di aggiornamenti locali.

Convergenza dell'iterazione dei valori

Come abbiamo detto, l'iterazione dei valori prima o poi converge a un insieme unico di soluzioni delle equazioni di Bellman. In questo sottoparagrafo spiegheremo perché ciò accade. Nel far questo introdurremo dei concetti matematici piuttosto utili e svilupperemo metodi per valutare l'errore nella funzione di utilità restituita quando l'algoritmo viene interrotto prematuramente; questo ci permette di non proseguire l'esecuzione all'infinito. Il contenuto di questo sottoparagrafo è piuttosto tecnico.

contrazione

Il concetto base che utilizzeremo per mostrare che l'iterazione dei valori converge è quello di **contrazione**. A grandi linee, una contrazione è una funzione a un argomento che, applicata successivamente a due diversi input, produce due valori di output che sono “più vi-

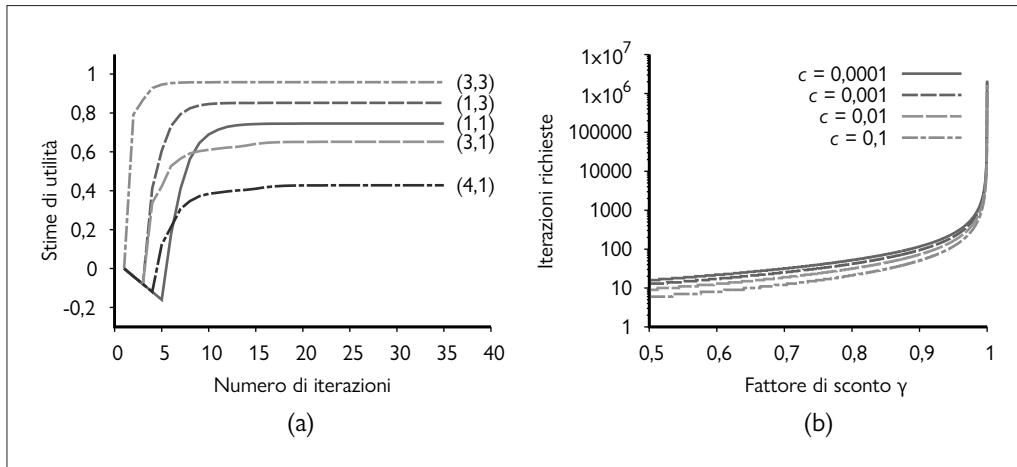


Figura 17.7 (a) Un grafico che riporta l’evoluzione delle utilità di alcuni stati durante l’iterazione dei valori. (b) Il numero di iterazioni richieste per garantire un errore massimo di $\varepsilon = c \cdot R_{\max}$, per valori diversi di c , in funzione del fattore di sconto γ .

cini”, almeno di un fattore costante, rispetto agli input originali. Per esempio, la funzione “divisione per due” è una contrazione, perché dopo che abbiamo diviso due numeri per due la loro differenza risulta dimezzata. Notate che la funzione “divisione per due” ha un punto fisso, che è zero, che non viene modificato dall’applicazione della funzione. Da quest’esempio possiamo estrapolare due proprietà importanti delle contrazioni.

- Una contrazione ha un solo punto fisso; infatti se ne avesse due, essi non potrebbero avvicinarsi quando la funzione viene loro applicata, quindi la funzione non sarebbe una contrazione.
- Quando la funzione è applicata a un qualsiasi argomento, il suo valore deve avvicinarsi al punto fisso (dato che quest’ultimo non si muove): quindi l’applicazione ripetuta di una contrazione tende sempre a raggiungere il punto fisso.

Ora supponiamo di considerare l’aggiornamento di Bellman (v. Equazione (17.10)) come un operatore B applicato simultaneamente per aggiornare l’utilità di ogni stato. Allora l’equazione di Bellman diventa $U = BU$ e l’equazione dell’aggiornamento di Bellman può essere scritta come:

$$U_{i+1} \leftarrow BU_i .$$

Ora ci serve un modo per misurare la distanza tra due vettori di utilità. Useremo la **norma del massimo**, che misura la “lunghezza” di un vettore con il valore assoluto del suo componente più grande:

$$\|U\| = \max_s \|U(s)\|.$$

Con questa definizione, la “distanza” tra due vettori $\|U - U'\|$ risulta pari alla differenza massima tra due elementi corrispondenti qualsiasi. Il risultato principale espresso in questo sottoparagrafo è il seguente: siano U_i e U'_i due vettori di utilità qualsiasi, allora risulta:

$$\|BU_i - BU'_i\| \leq \gamma \|U_i - U'_i\|. \quad (17.11)$$

Ovvero, l’aggiornamento di Bellman è una contrazione di un fattore γ sullo spazio dei vettori di utilità (l’Esercizio 17.VICT fornisce alcuni suggerimenti per la dimostrazione). Di conseguenza, dalle proprietà delle contrazioni in generale segue che l’iterazione dei valori converge sempre a una soluzione unica delle equazioni di Bellman ogni volta che $\gamma < 1$.



norma del massimo

Possiamo anche utilizzare la proprietà della contrazione per analizzare la *velocità* della convergenza a una soluzione. In particolare, possiamo sostituire U'_i nell'Equazione (17.11) con le *vere* utilità U , per cui $BU = U$. Otteniamo così la diseguaglianza:

$$\|BU_i - U\| \leq \gamma \|U_i - U\|.$$

Se consideriamo $\|U_i - U\|$ come l'*errore* nella stima U_i , vediamo che tale errore è ridotto di un fattore almeno pari a γ a ogni iterazione. Quindi, l'iterazione dei valori converge con velocità esponenziale. Possiamo calcolare il numero di iterazioni necessarie per raggiungere uno specifico margine d'errore ε come segue: innanzitutto ricordate dall'Equazione (17.1) che le utilità di tutti gli stati sono limitate da $\pm R_{\max}/(1 - \gamma)$. Questo significa che il massimo errore iniziale $\|U_0 - U\| \leq 2R_{\max}/(1 - \gamma)$. Supponiamo di eseguire N iterazioni per arrivare a un errore massimo di ε . Allora, dato che l'errore è ridotto di almeno γ ogni volta, è necessario che $\gamma^N \cdot 2R_{\max}/(1 - \gamma) \leq \varepsilon$. Passando ai logaritmi vediamo che:

$$N = \lceil \log(2R_{\max}/\varepsilon(1 - \gamma)) / \log(1/\gamma) \rceil$$

iterazioni sono sufficienti. La Figura 17.7(b) mostra come N varia con γ , per valori diversi del rapporto ε/R_{\max} . La buona notizia è che, grazie alla convergenza esponenzialmente rapida, N non dipende molto dal rapporto ε/R_{\max} . La cattiva notizia è che N cresce rapidamente man mano che γ si avvicina a 1. Possiamo ottenere una convergenza rapida se prendiamo un valore di γ piccolo, ma a tutti gli effetti questo significa imporre all'agente un orizzonte vicino, che potrebbe portarlo a non considerare gli effetti a lungo termine delle proprie azioni.

Il limite sull'errore discusso in precedenza dà un'idea dei fattori che influenzano il tempo di esecuzione dell'algoritmo, ma talvolta imporre un simile limite è un metodo troppo conservativo per decidere quando arrestare le iterazioni. Un soluzione alternativa consiste nel verificare le dimensioni dell'aggiornamento di Bellman in una data iterazione: dalla proprietà della contrazione (cfr. Equazione (17.11)) si può mostrare che, se l'aggiornamento è piccolo (cioè nessuna utilità cambia in modo significativo), allora anche l'errore dev'essere piccolo rispetto alla vera funzione di utilità. Più precisamente,

$$\text{se } \|U_{i+1} - U_i\| < \varepsilon(1 - \gamma)/\gamma \quad \text{allora } \|U_{i+1} - U\| < \varepsilon. \quad (17.12)$$

Questa è la condizione di terminazione dell'algoritmo ITERAZIONE-VALORI della Figura 17.6.

Fin qui abbiamo analizzato l'errore nella funzione di utilità restituita dall'algoritmo di iterazione dei valori. *Quello che interessa veramente all'agente, tuttavia, è quanto bene si comporterà qualora dovesse prendere le sue decisioni sulla base di tale funzione.* Supponiamo che dopo i iterazioni dell'algoritmo l'agente abbia una stima U_i della vera utilità U e ottenga la politica con la massima utilità attesa (MEU, *maximum expected utility*) π_i selezionando lo stato ottimo al passo successivo in base a U_i (come nell'Equazione (17.4)). Il comportamento risultante sarà vicino a quello ottimo? Si tratta di una questione fondamentale per qualsiasi agente reale, e per fortuna la risposta risulta essere affermativa. $U^{\pi_i}(s)$ è l'utilità ottenuta eseguendo π_i a partire da s , e la **perdita della politica** $\|U^{\pi_i} - U\|$ è la quantità massima di utilità che l'agente può perdere eseguendo π_i al posto della politica ottima π^* . La perdita di politica di π_i è collegata all'errore di U_i dalla seguente diseguaglianza:

$$\text{se } \|U_i - U\| < \varepsilon \quad \text{allora } \|U^{\pi_i} - U\| < 2\varepsilon. \quad (17.13)$$

In pratica si verifica spesso che π_i diventi ottima molto prima della convergenza di U_i . La Figura 17.8 mostra come l'errore massimo di U_i e la perdita della politica tendano a zero man mano che il processo di iterazione dei valori procede nell'ambiente 4×3 con $\gamma = 0,9$. La politica π_i è già ottima quando $i = 5$, sebbene l'errore massimo in U_i valga ancora 0,51.

Ora abbiamo tutto ciò che ci occorre per usare l'iterazione dei valori nella pratica. Sappiamo che converge alle utilità corrette, possiamo limitare l'errore nelle stime di utilità in un numero finito di iterazioni, e possiamo anche limitare la perdita risultante dall'esecuzione della politica MEU corrispondente. Come considerazione finale, notate che tutti i risultati



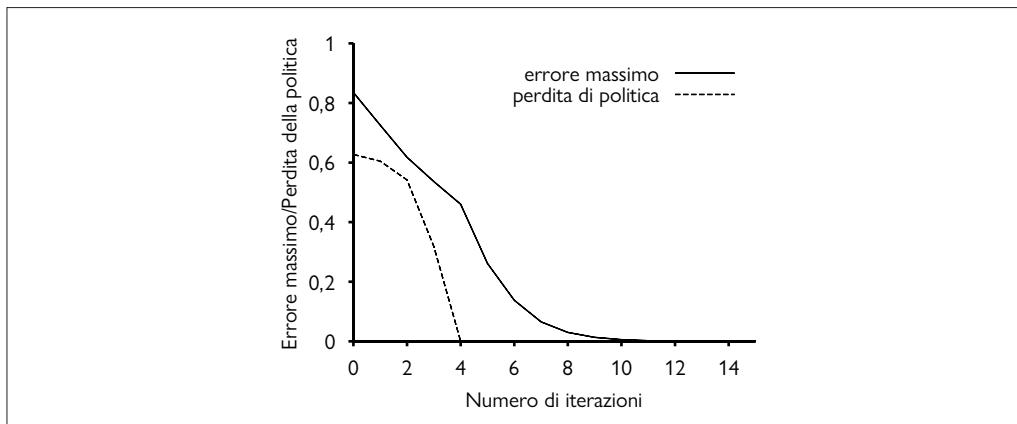


Figura 17.8
L'errore massimo $\|U_i - U\|$ delle stime di utilità e la perdita della politica $\|U^{pi} - U\|$ in funzione del numero di cicli dell'iterazione dei valori sul mondo 4×3 .

presentati in questo paragrafo dipendono dall'adozione di un fattore di sconto $\gamma < 1$. Se $\gamma = 1$ e l'ambiente contiene stati terminali, è possibile derivare un insieme analogo di risultati di convergenza e limitazione dell'errore.

17.2.2 Iterazione delle politiche

Nel paragrafo precedente abbiamo osservato che è possibile ottenere una politica ottima anche quando la stima della funzione di utilità non è del tutto accurata. Se un'azione è chiaramente preferibile a tutte le altre, l'esatto valore delle utilità degli stati coinvolti non deve necessariamente essere preciso. Quest'intuizione ci suggerisce un modo alternativo di trovare politiche ottime. L'algoritmo di **iterazione delle politiche** alterna i due passi qui sotto, partendo da una politica iniziale π_0 .

- **Valutazione della politica:** data una politica π_i , si calcola $U_i = U^{\pi_i}$, l'utilità di ogni stato qualora si dovesse eseguire π_i .
- **Miglioramento della politica:** si calcola una nuova politica MEU π_{i+1} , scegliendo gli stati successivi in base a U_i (come nell'Equazione (17.4)).

iterazione delle politiche

valutazione della politica

miglioramento della politica

L'algoritmo termina quando il passo di miglioramento della politica non apporta alcun cambiamento alle utilità: a questo punto sappiamo che la funzione di utilità U_i è un punto fisso dell'aggiornamento di Bellman, per cui dev'essere una soluzione delle equazioni di Bellman, e π_i sarà quindi una politica ottima. Dato che in uno spazio degli stati finito dev'essere finito anche il numero delle politiche, e si può dimostrare che ogni ciclo produce una politica migliore, l'iterazione delle politiche terminerà per forza. L'algoritmo è riportato nella Figura 17.9. Come per l'iterazione dei valori, utilizziamo la funzione Valore-Q definita in precedenza nel Paragrafo 17.1.2.

Come possiamo implementare la procedura VALUTAZIONE-POLITICA? È molto più facile fare questo che risolvere le equazioni di Bellman standard (che è ciò che fa l'iterazione dei valori), perché l'azione in ogni stato è fissata dalla politica. All' i -esima iterazione, la politica π_i nello stato s specifica l'azione $\pi_i(s)$. Questo significa che abbiamo una versione semplificata dell'equazione di Bellman (17.5) che collega l'utilità di s (data π_i) a quella dei suoi vicini:

$$U_i(s) = \sum_{s'} P(s' | s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')]. \quad (17.14)$$

Supponiamo per esempio che π_i sia la politica mostrata nella Figura 17.2(a). Allora abbiamo $\pi_i(1, 1) = Su$, $\pi_i(1, 2) = Su$ e così via, e le equazioni di Bellman semplificate sono:

$$\begin{aligned} U_i(1, 1) &= 0,8[-0,04 + U_i(1, 2)] + 0,1[-0,04 + U_i(2, 1)] + 0,1[-0,04 + U_i(1, 1)], \\ U_i(1, 2) &= 0,8[-0,04 + U_i(1, 3)] + 0,2[-0,04 + U_i(1, 2)], \end{aligned}$$

```

function ITERAZIONE-POLITICHE(mdp) returns una politica
  inputs: mdp, un MDP con stati S, azioni A(s), modello di transizione  $P(s'|s,a)$ 
  local variables: U, un vettore di utilità per gli stati in S, inizialmente a zero
     $\pi$ , un vettore rappresentante una politica e indicizzato per stato, inizialmente
    contiene casuali

  repeat
    U  $\leftarrow$  VALUTAZIONE-POLITICA( $\pi$ , U, mdp)
    immutata?  $\leftarrow$  true
    for each stato s in S do
       $a^* \leftarrow \underset{a \in A(s)}{\operatorname{argmax}} \text{VALORE-Q}(\textit{mdp}, s, a, U)$ 
      if  $\text{VALORE-Q}(\textit{mdp}, s, a^*, U) > \text{VALORE-Q}(\textit{mdp}, s, \pi[s], U)$  then
         $\pi[s] \leftarrow a^*$ ; immutata?  $\leftarrow$  false
    until immutata?
    return  $\pi$ 

```

Figura 17.9 L'algoritmo di iterazione delle politiche per il calcolo della politica ottima.

e così via per tutti gli stati. L'aspetto fondamentale di queste equazioni è che sono *lineari*, perché l'operatore “max” è stato rimosso. Per n stati, abbiamo n equazioni lineari in n incognite, che possono essere risolte in modo esatto in un tempo $O(n^3)$ applicando metodi algebrici standard. Se il modello di transizione è sparso – cioè se da ogni stato si passa soltanto in un piccolo numero di altri stati – allora il processo di soluzione può essere ancora più veloce.

Quando gli spazi degli stati sono piccoli, la valutazione delle politiche con metodi esatti di risoluzione è spesso l'approccio più efficiente. Quando gli spazi degli stati diventano più grandi, un tempo $O(n^3)$ potrebbe risultare proibitivo. Fortunatamente, non è necessario valutare le politiche in modo *esatto*; possiamo invece eseguire un certo numero di passi di iterazione dei valori (semplificati, perché la politica è fissata) per ottenere un'approssimazione delle utilità ragionevolmente buona. L'aggiornamento di Bellman semplificato per questo processo è:

$$U_{i+1}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')],$$

e viene ripetuto diverse volte per produrre in modo efficiente la stima di utilità. L'algoritmo risultante è chiamato **iterazione delle politiche modificata**.

iterazione delle politiche modificata

Gli algoritmi descritti sin qui richiedono di aggiornare l'utilità o la politica per tutti gli stati nello stesso momento. Questo non è strettamente necessario: in effetti, a ogni iterazione possiamo scegliere *un sottoinsieme qualsiasi* degli stati e applicare a esso *l'uno o l'altro* degli aggiornamenti (miglioramento della politica o iterazione dei valori semplificata). Quest'algoritmo molto generale è chiamato **iterazione asincrona delle politiche**. Date certe condizioni sulla politica iniziale e la funzione di utilità iniziale, è garantito che l'iterazione asincrona delle politiche converga a una politica ottima. La libertà di scegliere gli stati su cui lavorare significa che è possibile progettare algoritmi euristici molto più efficienti: per esempio, algoritmi che si concentrano sull'aggiornamento dei valori degli stati che è molto probabile raggiungere applicando una buona politica. Non ha senso pianificare per i risultati di un'azione che non sarà mai eseguita.

iterazione asincrona delle politiche

17.2.3 Programmazione lineare

La **programmazione lineare** o PL, accennata brevemente nel Capitolo 4 (Paragrafo 4.2), è un approccio generale per formulare problemi di ottimizzazione vincolata; sono disponibili molti risolutori PL di livello industriale. Dato che le equazioni di Bellman prevedono molte operazioni di somma e calcolo del massimo, non sorprende che la risoluzione di un problema MDP si possa ridurre alla risoluzione di un programma lineare opportunamente formulato.

L'idea fondamentale della formulazione consiste nel considerare come variabili nella PL le utilità $U(s)$ di ogni stato s , osservando che le utilità per una politica ottima sono le massime utilità ottenibili che siano consistenti con le equazioni di Bellman. Nella terminologia della PL, questo significa che cerchiamo di minimizzare $U(s)$ per tutti gli s che soddisfano le diseguaglianze:

$$U(s) \geq \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s')]$$

per ogni stato s e ogni azione a .

Si crea così una connessione tra la programmazione dinamica e la programmazione lineare, per cui sono stati studiati in modo molto approfondito gli algoritmi e le questioni di complessità. Per esempio, grazie al fatto che la programmazione lineare è risolvibile in tempo polinomiale, si può mostrare che gli MDP possono essere risolti in tempo polinomiale nel numero di stati e azioni e nel numero di bit richiesti per specificare il modello. Nella pratica si verifica che i risolutori PL solo raramente sono efficienti quanto la programmazione dinamica, per la risoluzione di MDP. Inoltre, il tempo polinomiale sembra un vantaggio, ma il numero di stati spesso è molto grande. Infine, occorre ricordare che anche il più semplice e il meno informato degli algoritmi di ricerca del Capitolo 3 si esegue in tempo lineare nel numero di stati e azioni.

17.2.4 Algoritmi online per MDP

L'iterazione dei valori e l'iterazione delle politiche sono algoritmi *offline*: come l'algoritmo A* del Capitolo 3, generano una soluzione ottima per il problema, che può poi essere eseguita da un semplice agente. Per MDP sufficientemente grandi, come quello del Tetris con 10^{62} stati, ottenere una soluzione offline esatta, anche mediante un algoritmo in tempo polinomiale, non è possibile. Sono state sviluppate diverse tecniche per ottenere soluzioni offline approssimate di problemi MDP; tali tecniche sono trattate nelle note storiche e bibliografiche al termine di questo capitolo e nel Capitolo 22 del Volume 2.

Noi qui prenderemo in esame algoritmi online, analoghi a quelli usati per i giochi nel Capitolo 5, in cui l'agente svolge una notevole quantità di calcoli in ogni punto decisionale, anziché operare principalmente con informazioni precalcolate.

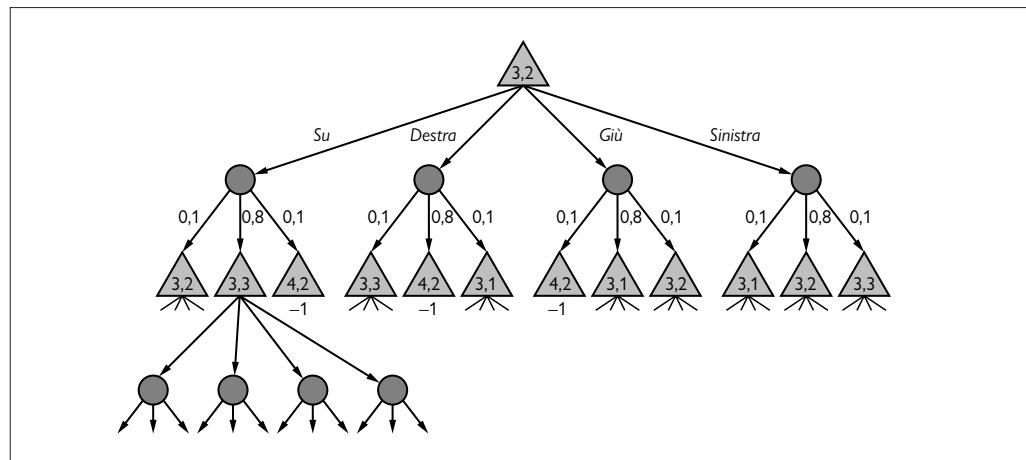
L'approccio più diretto è in effetti una semplificazione dell'algoritmo EXPECTIMINIMAX per gli alberi di gioco con nodi di casualità: l'algoritmo EXPECTIMAX genera un albero con un'alternanza di nodi di massimo e nodi di casualità, come illustrato nella Figura 17.10 (c'è una leggera differenza rispetto all'algoritmo EXPECTIMINIMAX standard, perché ci sono ricompense anche su transazioni verso stati non terminali, oltre che su transazioni verso stati terminali). È possibile applicare una funzione di valutazione alle foglie non terminali dell'albero, oppure assegnare loro un valore di default. Si può estrarre una decisione dall'albero di ricerca portando su valori di utilità dalle foglie, effettuando una media nei nodi di casualità e considerando il massimo nei nodi decisionali.

Per problemi in cui il fattore di sconto γ non è troppo vicino a 1, è utile il concetto di ε -orizzonte. Sia ε un limite desiderato sull'errore assoluto delle utilità calcolate da un albero expectimax di profondità limitata, confrontate con le utilità esatte nell'MDP. Allora l' ε -orizzonte è la profondità dell'albero H tale che la somma delle ricompense al di là di qualsiasi

Figura 17.10

Una parte di un albero EXPECTIMAX per il problema MDP 4×3 con radice in (3,2).

I nodi triangolari sono nodi di massimo e quelli a forme di cerchio sono nodi di casualità.



foglia a tale livello di profondità è minore di ε – per dirla in parole semplici, qualsiasi cosa accada dopo H è irrilevante perché avviene in un futuro molto lontano. Poiché la somma delle ricompense al di là di H è limitata da $\gamma^H R_{\max}/(1-\gamma)$, è sufficiente una profondità $H = \lceil \log_\gamma \varepsilon(1-\gamma)/R_{\max} \rceil$. Quindi, costruendo un albero a questo livello di profondità si ottengono decisioni quasi ottime. Per esempio, con $\gamma = 0,5$, $\varepsilon = 0,1$ e $R_{\max} = 1$, troviamo $H = 5$, che sembra un valore ragionevole. D'altra parte, se $\gamma = 0,9$, $H = 44$, per nulla ragionevole!

Oltre a limitare la profondità, è anche possibile evitare il potenzialmente enorme fattore di ramificazione nei nodi di casualità (per esempio, se tutte le probabilità condizionate in un modello di transizione DBN sono non nulle, le probabilità di transizione, date dal prodotto delle probabilità condizionate, sono anch'esse non nulle, a indicare che da ogni stato c'è qualche probabilità di passare in ogni altro stato).

Come si è osservato nel Paragrafo 13.4, le aspettative rispetto a una distribuzione di probabilità P possono essere approssimate generando N campioni da P e usando la media campionaria. In termini matematici, abbiamo:

$$\sum_x P(x)f(x) \approx \frac{1}{N} \sum_{i=1}^N f(x_i).$$

Quindi, se il fattore di ramificazione è molto grande, a indicare che ci sono molti possibili valori di x , si può ottenere una buona approssimazione del valore del nodo di casualità campionando un numero limitato di risultati ottenuti eseguendo l'azione. Generalmente i campioni si focalizzeranno sui risultati più probabili, perché sono quelli che tendono ad avere più probabilità di essere generati.

Se osservate attentamente l'albero della Figura 17.10, noterete che in realtà non è proprio un albero. Per esempio, la radice (3,2) è anche una foglia, perciò dovremmo considerarlo un grafo, e imporre che il valore della foglia (3,2) coincida con il valore della radice (3,2), perché sono lo stesso stato. In effetti questa linea di pensiero ci riporta rapidamente alle equazioni di Bellman che mettono in relazione i valori degli stati ai valori dei vicini. Gli stati esplorati costituiscono in effetti un sotto-MDP dell'MDP originale, e questo sotto-MDP può essere risolto usando uno qualsiasi degli algoritmi trattati in questo capitolo per ottenere una decisione per lo stato corrente (agli stati di frontiera si assegna generalmente un valore stimato fissato).

Questo approccio generale si chiama **programmazione dinamica in tempo reale** (RTDP, *real-time dynamic programming*) ed è abbastanza simile all'approccio LRTA* del Capitolo 4. Gli algoritmi di questo tipo possono essere piuttosto efficaci in domini di dimensioni mo-

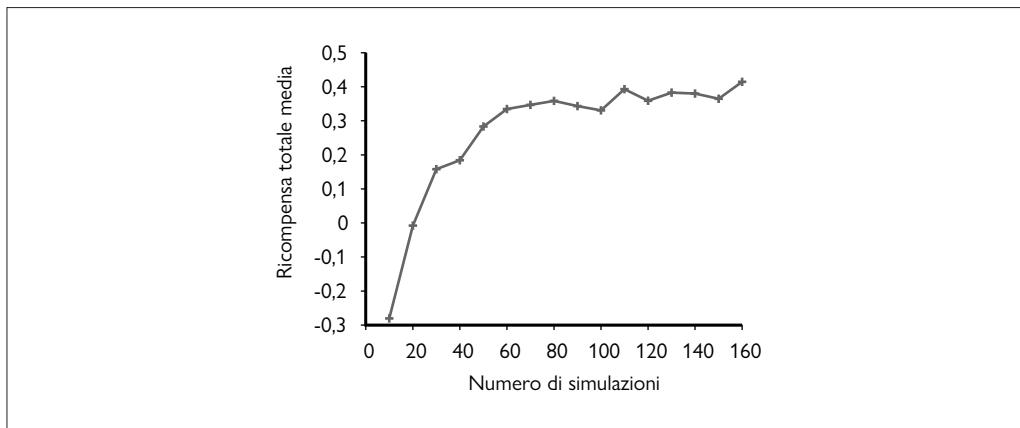


Figura 17.11
Prestazione della politica UCT in funzione del numero di simulazioni per mossa nel mondo 4×3 , usando una politica di simulazione casuale, come media calcolata su 1000 esecuzioni per ogni punto.

derate come i mondi a griglia, mentre in domini più grandi, come quello del Tetris, ci sono due questioni da affrontare.

In primo luogo, lo spazio degli stati è tale che qualsiasi insieme gestibile di stati esplorati contiene pochissimi stati ripetuti, perciò si potrebbe usare anche un semplice albero expectimax. In secondo luogo, una semplice euristica per i nodi di frontiera potrebbe non essere sufficiente a guidare l'agente, in particolare se le ricompense sono sparse.

Un possibile rimedio consiste nell'applicare l'apprendimento per rinforzo in modo da generare un'euristica molto più accurata (cfr. il Capitolo 22 del Volume 2). Un altro approccio è quello di guardare più in avanti nell'MDP usando il metodo Monte Carlo descritto nel Paragrafo 5.4. In effetti, l'algoritmo UCT della Figura 5.10 fu sviluppato in origine proprio per gli MDP e non per i giochi. I cambiamenti richiesti per risolvere MDP anziché giochi sono minimi: si devono principalmente al fatto che l'avversario (la natura) è stocastico e alla necessità di tenere traccia di ricompense e non solo di vincite e perdite.

Tuttavia, applicando l'UCT al mondo 4×3 si ottengono prestazioni non particolarmente entusiasmanti. Come si vede nella Figura 17.11, servono in media 160 simulazioni per raggiungere una ricompensa totale di 0,4, mentre una politica ottima ha una ricompensa totale attesa di 0,7453 partendo dallo stato iniziale (Figura 17.3). Un motivo che ostacola il compito dell'UCT è che tale algoritmo costruisce un albero anziché un grafo e utilizza (un'approssimazione di) expectimax anziché la programmazione dinamica. Il mondo 4×3 è molto "tor tuoso": benché vi siano soltanto 9 stati non terminali, le simulazioni UCT spesso continuano per più di 50 azioni.

L'UCT sembra più adatto al Tetris, dove le simulazioni vanno abbastanza in avanti da consentire all'agente di capire se una mossa potenzialmente rischiosa alla fine funzionerà oppure causerà un eccessivo impilamento. L'Esercizio 17.UCTT esamina l'applicazione dell'UCT al Tetris. Una questione particolarmente interessante riguarda quanto possa essere d'aiuto una politica di simulazione semplice, per esempio una che eviti di creare sbalzi eccessivi e inserisca i pezzi più in basso possibile.

17.3 Problemi dei banditi

A Las Vegas, un *bandito con un braccio solo* indica una slot machine. Un giocatore può inserire un gettone, tirare la leva e raccogliere la vincita (se c'è). Un bandito con **n braccia** ha **n** leve, per ognuna delle quali c'è una distribuzione di probabilità fissata, ma ignota, di vincite: ogni volta che si tira una leva si esegue un campionamento dalla distribuzione ignota corrispondente.

**bandito con
n braccia**

Il giocatore deve scegliere quale leva usare ogni volta che inserisce un altro gettone: quella che finora ha pagato di più, o magari quella che non ha ancora provato? Questo è un esempio del bilanciamento tra **sfruttamento** della migliore azione attuale per ottenere ricompense ed **esplorazione** di stati e azioni precedentemente ignoti per ottenere informazioni, che in alcuni casi possono condurre a una politica migliore e a maggiori ricompense nel lungo periodo. Nel mondo reale occorre sempre decidere tra continuare a muoversi in una tranquilla zona di comfort e lanciarsi a esplorare l'ignoto nella speranza di una vita migliore.

Il problema del bandito con n braccia è un modello formale per problemi reali in molti campi di importanza vitale, come decidere quali di n possibili nuove cure usare per il trattamento di una malattia, quali di n possibili investimenti scegliere per investire parte dei propri risparmi, quali di n possibili progetti di ricerca finanziare, o quali di n possibili messaggi pubblicitari visualizzare quando un utente visita una particolare pagina web.

I primi lavori su questo problema furono condotti negli Stati Uniti durante la seconda guerra mondiale, e si rivelarono talmente complessi che gli scienziati alleati proposero di “affidare il problema alla Germania, come strumento finale di sabotaggio intellettuale” (Whittle, 1979).

Gli scienziati, durante e dopo il periodo della guerra, cercavano di dimostrare che fatti “ovviamente veri” sui problemi dei banditi erano, in realtà, falsi (nelle parole di Bradt *et al.* (1956): “Ci sono molte proprietà interessanti che le strategie ottime non possiedono”). Per esempio, in generale si dava per scontato che una politica ottima alla fine sarebbe arrivata a indicare il braccio migliore nel lungo periodo; ma in realtà c’è una probabilità finita che una politica ottima indichi un braccio subottimo. Oggi disponiamo di una solida conoscenza teorica dei problemi dei banditi, nonché di utili algoritmi per risolverli.

problema dei banditi

Esistono diverse definizioni di **problemi dei banditi**; una delle più pulite e più generali è la seguente.

processo di ricompensa di Markov

- Ogni braccio M_i è un **processo di ricompensa di Markov** (MRP, *Markov reward process*), cioè un MDP con una sola azione possibile, a_i e stati S_i , modello di transizione $P_i(s' | s, a_i)$ e ricompensa $R_i(s, a_i, s')$. Il braccio definisce una distribuzione su sequenze di ricompense $R_{i,0}, R_{i,1}, R_{i,2}, \dots$, dove ogni $R_{i,t}$ è una variabile casuale.
- Il problema dei banditi complessivo è un MDP: lo spazio degli stati è dato dal prodotto cartesiano $S = S_1 \times \dots \times S_n$; le azioni sono a_1, \dots, a_n ; il modello di transizione aggiorna lo stato del braccio M_i selezionato, secondo il suo specifico modello di transizione, lasciando invariate le altre braccia; il fattore di sconto è γ .

Questa definizione è molto generale e copre un’ampia varietà di casi. La proprietà fondamentale è l’indipendenza delle braccia, insieme al fatto che l’agente può lavorare su un solo braccio per volta. È possibile definire una versione ancora più generale in cui si possono applicare sforzi frazionari a tutte le braccia simultaneamente, ma il lavoro totale su tutte le braccia è limitato; i risultati di base descritti qui coprono anche questo caso.

Vedremo tra breve come formulare un tipico problema dei banditi nel quadro di riferimento qui definito, ma cominciamo con il semplice caso particolare di un problema con sequenze di ricompense deterministiche. Sia $\gamma = 0,5$ e supponiamo che vi siano due braccia etichettate M e M_1 . Tirando M più volte si ottiene la sequenza di ricompense 0, 2, 0, 7, 2, 0, 0, ..., mentre tirando M_1 si ottiene 1, 1, 1, ... (Figura 17.12(a)). Se all’inizio un agente dovesse scegliere un braccio o l’altro e mantenere la scelta, la scelta andrebbe fatta calcolando l’utilità (ricompensa totale scontata) per ogni braccio:

$$U(M) = (1,0 \times 0) + (0,5 \times 2) + (0,5^2 \times 0) + (0,5^3 \times 7,2) = 1,9$$

$$U(M_1) = \sum_{t=0}^{\infty} 0,5^t = 2,0.$$

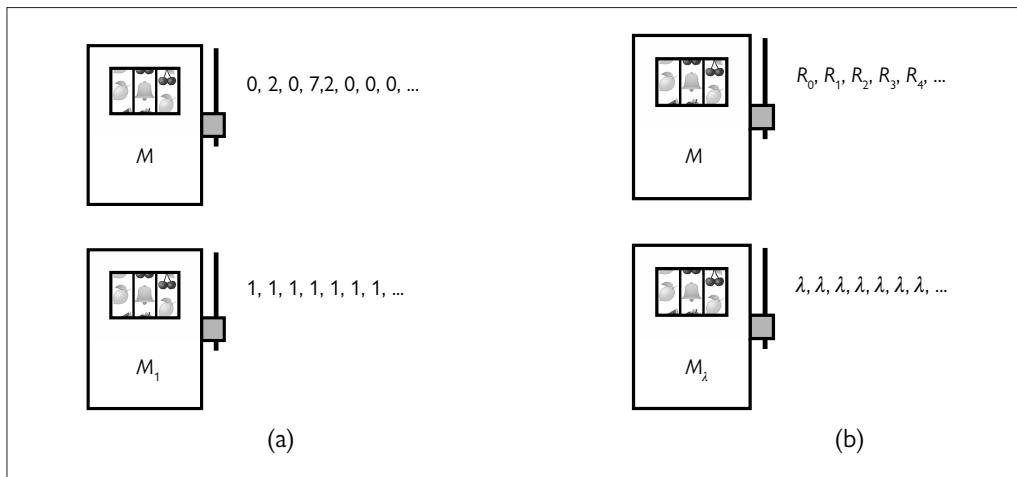


Figura 17.12 (a) Un semplice problema dei banditi deterministico con due braccia: le braccia possono essere tirate in qualsiasi ordine, e ognuna di esse genera la sequenza di ricompense mostrata in figura. (b) Un caso più generale del problema dei banditi in (a), in cui il primo braccio fornisce una sequenza arbitraria di ricompense e il secondo fornisce una ricompensa fissata λ .

Si potrebbe pensare che la scelta migliore sia M_1 , ma riflettendo meglio si vede che iniziando con M e passando poi a M_1 dopo la quarta ricompensa si ottiene la sequenza $S = 0, 2, 0, 7, 2, 1, 1, 1, \dots$, per cui:

$$U(S) = (1, 0 \times 0) + (0, 5 \times 2) + (0, 5^2 \times 0) + (0, 5^3 \times 7, 2) + \sum_{t=4}^{\infty} 0, 5^t = 2, 025.$$

Quindi la strategia S che passa da M a M_1 al momento giusto è migliore della scelta di una delle due braccia senza cambiare. In effetti, S è ottima per questo problema: scegliendo un altro tempo per il cambio del braccio, la ricompensa sarebbe minore.

Ora generalizziamo questo caso, in modo che il primo braccio M generi una sequenza arbitraria R_0, R_1, R_2, \dots (che può essere nota o meno) e il secondo braccio M_λ generi la sequenza $\lambda, \lambda, \lambda, \dots$ per una costante fissata nota λ (cfr. Figura 17.12(b)). In letteratura si parla in questo caso di problema del **bandito con un braccio solo**, perché è formalmente equivalente al caso in cui c'è un solo braccio M che produce le ricompense R_0, R_1, R_2, \dots e i costi λ a ogni tiro (tirare il braccio M è equivalente a non tirare il braccio M_λ , perciò rinuncia a una ricompensa di λ ogni volta). Con un solo braccio, l'unica scelta è tra tirare ancora o fermarsi. Se si tira il primo braccio per T volte (cioè ai tempi $0, 1, \dots, T-1$, si dice che il **tempo di arresto** è T .

Tornando alla nostra versione con braccia M e M_λ , ipotizziamo che dopo T tiri del primo braccio, una strategia ottima preveda di tirare il secondo braccio per la prima volta. Poiché questa mossa non fornisce ulteriori informazioni (sappiamo già che il payoff sarà λ), al tempo $T + 1$ saremo nella stessa situazione, quindi una strategia ottima deve fare la stessa scelta.

In modo equivalente, possiamo dire che una strategia ottima è quella di tirare il braccio M fino al tempo T e poi passare al braccio M_λ per il tempo che rimane. È possibile che $T = 0$ se la strategia sceglie M_λ immediatamente, o $T = \infty$ se la strategia non sceglie mai M_λ , o un valore compreso tra i due. Ora consideriamo il valore di λ tale che una strategia ottima sia *esattamente indifferente* tra (a) usare M fino al miglior tempo di arresto possibile e poi passare a M_λ per sempre, e (b) scegliere M_λ immediatamente. Al punto di svolta abbiamo:

$$\max_{T>0} E \left[\left(\sum_{t=0}^{T-1} \gamma^t R_t \right) + \sum_{t=T}^{\infty} \gamma^t \lambda \right] = \sum_{t=0}^{\infty} \gamma^t \lambda,$$

**bandito con
un braccio solo**

tempo di arresto

che si semplifica in:

$$\lambda = \max_{T>0} \frac{E(\sum_{t=0}^{T-1} \gamma^t R_t)}{E(\sum_{t=0}^{T-1} \gamma^t)}. \quad (17.15)$$

Questa equazione definisce un tipo di “valore” per M in termini della sua capacità di fornire un flusso di ricompense puntuale; il numeratore della frazione rappresenta un’utile, mentre il denominatore può essere visto come un “tempo scontato”, per cui il valore descrive la massima utile ottenibile per unità di tempo scontato (è importante ricordare che T nell’equazione è un tempo di arresto, che non è un semplice intero ma è governato da una regola per arrestare l’utilizzo del primo braccio; si riduce a un semplice intero soltanto quando M è una sequenza di ricompense deterministica). Il valore definito nell’Equazione (17.15) è chiamato **indice di Gittins** di M .

indice di Gittins



L’importanza dell’indice di Gittins si deve al fatto che fornisce una politica ottima molto semplice per qualsiasi problema dei banditi: *tirare il braccio che ha l’indice di Gittins più alto, poi aggiornare gli indici di Gittins*. Inoltre, poiché l’indice di Gittins del braccio M_i dipende solo dalle proprietà di tale braccio, una decisione ottima sulla prima iterazione si può calcolare in tempo $O(n)$, dove n è il numero di braccia. E poiché gli indici di Gittins delle braccia non selezionate rimangono invariati, ogni decisione dopo la prima può essere calcolata in tempo $O(1)$.

17.3.1 Calcolo dell’indice di Gittins

Per capire meglio il significato dell’indice di Gittins, calcoliamo il valore del numeratore, del denominatore e del rapporto nell’Equazione (17.15) per diversi possibili tempi di arresto sulla sequenza di ricompense deterministica $0, 2, 0, 7, 2, 0, 0, 0, \dots$:

T	1	2	3	4	5	6
R_t	0	2	0	7,2	0	0
$\Sigma \gamma^t R_t$	0,0	1,0	1,0	1,9	1,9	1,9
$\Sigma \gamma^t$	1,0	1,5	1,75	1,875	1,9375	1,9687
rapporto	0,0	0,6667	0,5714	1,0133	0,9806	0,9651

Risulta chiaro che il rapporto diminuirà da qui in poi, perché il numeratore rimane costante mentre il denominatore continua ad aumentare. Quindi, l’indice di Gittins per questo braccio è 1,0133, il valore massimo raggiunto dal rapporto. Per un braccio fissato M_λ con $0 < \lambda \leq 1,0133$, la politica ottima raccoglie le prime quattro ricompense da M e poi passa a M_λ . Per $\lambda > 1,0133$, la politica ottima sceglie sempre M_λ .

Per calcolare l’indice di Gittins per un braccio generico M con stato corrente s , basta fare la seguente osservazione: nel punto di svolta in cui una politica ottima è indifferente tra scegliere il braccio M e scegliere il braccio fissato M_λ , il valore di scegliere M è identico a quello di scegliere una sequenza infinita di ricompense λ .

Supponiamo di aumentare M in modo che in ogni suo stato l’agente abbia due scelte a disposizione: continuare con M come prima, oppure abbandonare e ricevere una sequenza infinita di ricompense λ (Figura 17.13(a)). Questo trasforma M in un MDP la cui politica ottima è semplicemente la regola di arresto ottimo per M . Quindi, il valore di una politica ottima in questo nuovo MDP è uguale al valore di una sequenza infinita di ricompense λ , cioè $\lambda/(1 - \gamma)$. Perciò possiamo risolvere questo MDP... ma sfortunatamente non conosciamo il valore di λ da inserire nel problema, dato che è proprio quello che stiamo cercando di calcolare. Tuttavia, sappiamo che nel punto di svolta una politica ottima è indifferente tra M e M_λ , perciò possiamo sostituire la scelta di ottenere una sequenza infinita di ricompense λ

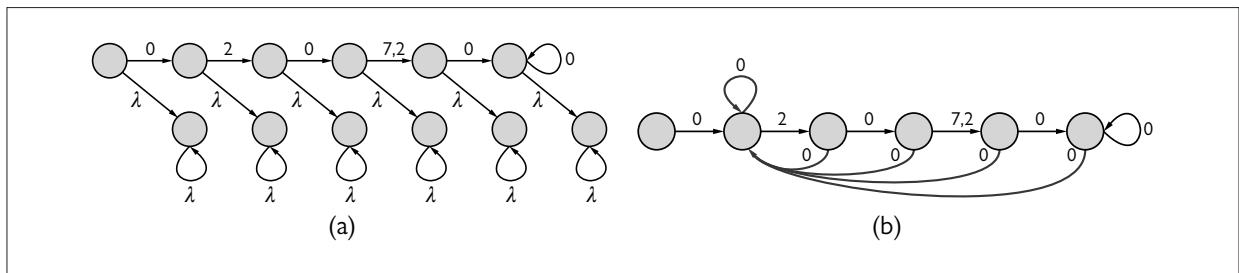


Figura 17.13 (a) La sequenza di ricompense $M = 0, 2, 0, 7.2, 0, 0, 0, \dots$ aumentata con la possibilità di passare definitivamente a un braccio costante M_λ in ogni istante temporale. (b) Un MDP il cui valore ottimo è esattamente equivalente al valore ottimo per (a), nel punto in cui la politica ottima è indifferente tra M e M_λ .

con la scelta di tornare indietro e ripartire con M dal suo stato iniziale s (più precisamente, aggiungiamo una nuova azione in ogni stato che ha le stesse ricompense e gli stessi esiti dell'azione disponibile in s ; cfr. l'Esercizio 17.KATV). Questo nuovo MDP, M^s , detto **MDP di ripartenza**,

MDP di ripartenza

Abbiamo così ottenuto il risultato generale che l'indice di Gittins per un braccio M nello stato s è uguale a $1 - \gamma$ volte il valore di una politica ottima per l'MDP di ripartenza M^s . Questo MDP può essere risolto da uno qualsiasi degli algoritmi descritti nel Paragrafo 17.2. L'iterazione dei valori applicata a M^s nella Figura 17.13(b) fornisce un valore di 2,0266 per lo stato iniziale, perciò abbiamo $\lambda = 2,0266 (1 - \gamma) = 1,0133$ come prima.

17.3.2 Il problema dei banditi di Bernoulli

L'esempio forse più semplice e più noto di problema dei banditi è il **problema dei banditi di Bernoulli**, in cui ogni braccio M_i produce una ricompensa di 0 o 1 con una probabilità fissa ma ignota μ_i . Lo stato del braccio M_i è definito da s_i e f_i , i conteggi di successi (1) e fallimenti (0) ottenuti finora per quel braccio; la probabilità di transizione predice che il prossimo esito sarà 1 con probabilità $(s_i)/(s_i + f_i)$ e 0 con probabilità $(f_i)/(s_i + f_i)$. I conteggi sono inizializzati a 1, per cui le probabilità iniziali sono 1/2 anziché 0/0.⁴ Il processo di ricompensa di Markov è mostrato nella Figura 17.14(a).

**problema dei
banditi di Bernoulli**

Non possiamo applicare la trasformazione del paragrafo precedente per calcolare l'indice di Gittins del braccio di Bernoulli, perché ha un numero di stati infinito. Possiamo però ottenere un'approssimazione molto accurata risolvendo l'MDP troncato con stati fino a $s_i + f_i = 100$ e $\gamma = 0,9$. I risultati sono mostrati nella Figura 17.14(b) e appaiono intuitivamente ragionevoli: vediamo che, in termini generali, vengono preferite le braccia con probabilità di payoff più elevate, ma c'è anche un **premio di esplorazione** per le braccia che sono state sperimentate poche volte. Per esempio, l'indice di Gittins per lo stato (3,2) è maggiore dell'indice per lo stato (7,4) (0,7057 contro 0,6922), anche se il valore stimato in (3,2) è minore (0,6 contro 0,6364).

**premio di
esplorazione**

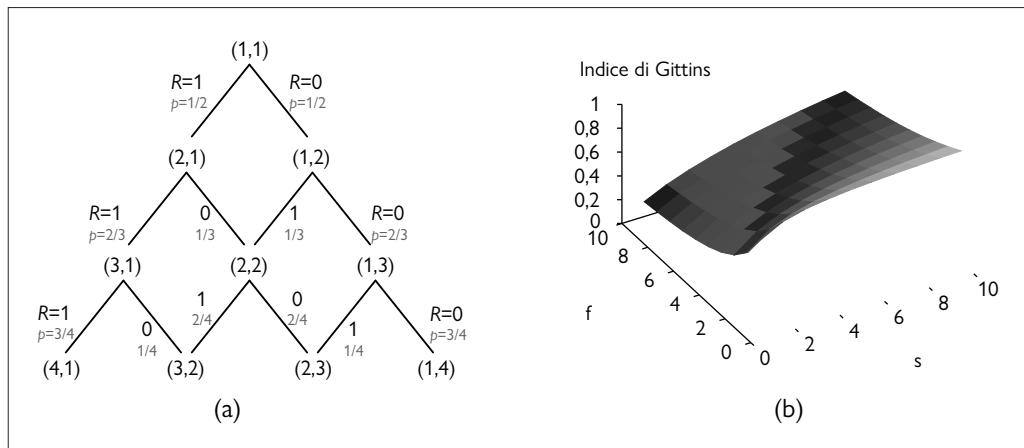
17.3.3 Politiche quasi ottime per problemi dei banditi

Calcolare gli indici di Gittins per problemi realistici non è quasi mai facile. Fortunatamente, le proprietà generali osservate nel paragrafo precedente, ovvero la desiderabilità di alcune combinazioni di valore stimato e incertezza, portano di per sé alla creazione di politiche semplici che risultano “quasi altrettanto valide” delle politiche ottime.

⁴ Le probabilità sono quelle di un processo di aggiornamento bayesiano con una distribuzione a priori Beta(1,1) (cfr. Paragrafo 20.2.5 del Volume 2).

Figura 17.14

(a) Stati, ricompense e probabilità di transizione per il problema dei banditi di Bernoulli. (b) Indici di Gittins per gli stati del processo per il problema dei banditi di Bernoulli.



limite superiore di confidenza

La prima classe di metodi utilizza l’euristica dei **limite superiore di confidenza** o UCB (*upper confidence bound*), già introdotta nel Capitolo 5 per la ricerca ad albero di Monte Carlo (Figura 5.11). L’idea di base è quella di usare i campioni tratti da ciascun braccio per stabilire un **intervallo di confidenza** per il valore del braccio in questione, ovvero un intervallo entro il quale si può stimare – con alto grado di confidenza – che si trovi il valore; e poi scegliere il braccio con il limite superiore più alto sul suo intervallo di confidenza. Il limite superiore è dato dalla stima del valore medio corrente $\hat{\mu}_i$ più un multiplo della deviazione standard dell’incertezza nel valore. La deviazione standard è proporzionale a $\sqrt{1/N_i}$, dove N_i è il numero di volte che è stato campionato il braccio M_i . Abbiamo così un indice approssimato del valore per il braccio M_i dato da:

$$UCB(M_i) = \hat{\mu}_i + g(N)/\sqrt{N_i},$$

dove $g(N)$ è una funzione opportunamente scelta di N , il numero totale di campioni ottenuti da tutte le braccia. Una politica UCB non fa altro che scegliere il braccio con il valore UCB più alto. Notate che il valore UCB non è strettamente un indice, perché dipende da N , il numero totale di campioni ottenuti da tutte le braccia, e non solo dal particolare braccio considerato.

La definizione precisa di g determina il **rimpianto** (*regret*) relativo alla politica del veggenti, che non fa altro che scegliere il braccio migliore e ottenere una ricompensa media μ^* . Un famoso risultato ottenuto da Lai e Robbins (1985) mostra che, nel caso non scontato, nessun algoritmo può avere un rimpianto che cresce più lentamente di $O(\log N)$. Diverse scelte di g portano a una politica UCB che corrisponde a questa crescita; per esempio, possiamo usare $g(N) = (2 \log(1 + N \log^2 N))^{1/2}$.

campionamento di Thompson

Un altro metodo, il **campionamento di Thompson** (Thompson, 1933), sceglie un braccio in base alla probabilità che esso sia ottimo dati i campioni ottenuti finora. Supponiamo che $P_i(\mu_i)$ sia la distribuzione di probabilità corrente per il vero valore del braccio M_i . Allora un modo semplice per implementare il campionamento di Thompson consiste nel generare un campione da ogni P_i e poi scegliere il campione migliore. Questo algoritmo ha un rimpianto che cresce come $O(\log N)$.

17.3.4 Varianti non indicizzabili

I problemi dei banditi sono nati in parte a seguito del compito di testare nuove cure mediche su pazienti gravemente malati. Per questo compito, lo scopo di massimizzare il numero totale di successi nel tempo è evidentemente sensato: ogni test di successo significa una vita salvata, ogni fallimento una vita perduta.

Tuttavia, se cambiamo leggermente le ipotesi, emerge un problema diverso. Supponiamo che, anziché determinare la migliore cura medica per ogni nuovo paziente umano, procediamo a testare diversi farmaci su campioni di batteri allo scopo di decidere quale sia il farmaco migliore. In seguito metteremo in commercio quel farmaco e dimenticheremo gli altri. In questo scenario non ci sono costi aggiuntivi se i batteri muoiono, c'è un costo fisso per ogni test, ma non dobbiamo ridurre al minimo i fallimenti dei test; stiamo semplicemente cercando di prendere una buona decisione nel modo più rapido possibile.

Il compito di scegliere l'opzione migliore in queste condizioni è chiamato **problema di selezione**. I problemi di selezione si incontrano ovunque in contesti industriali e personali. Spesso occorre decidere quale fornitore usare per un processo; o quale candidato assumere per un impiego. A prima vista i problemi di selezione appaiono simili al problema dei banditi, ma in realtà possono avere proprietà matematiche diverse. In particolare, *non esiste alcuna funzione indice per problemi di selezione*. Per dimostrarlo occorre mostrare un qualunque scenario in cui la politica ottima cambia le sue preferenze per due braccia M_1 e M_2 quando si aggiunge un terzo braccio M_3 (cfr. l'Esercizio 17.SELC).

problema
di selezione



Nel Capitolo 5 si è introdotto il concetto di problemi decisionali di **metalivello**, come quello di decidere quali calcoli effettuare, durante una ricerca su un albero di gioco, prima di fare una mossa. Una decisione di metalivello di questo tipo è un problema di selezione più che un problema dei banditi. Chiaramente l'espansione o la valutazione di un nodo *costa* la stessa quantità di tempo a prescindere dal fatto che produca un valore alto o basso. Risulta forse sorprendente, quindi, che l'algoritmo di ricerca ad albero Monte Carlo (cfr. Paragrafo 5.4) abbia riscosso successo, dato che cerca di risolvere problemi di selezione con l'euristica UCB, progettata per i problemi dei banditi. In termini generali, ci si aspetterebbe che gli algoritmi ottimi per i problemi dei banditi esplorino molto meno degli algoritmi ottimi per i problemi di selezione, poiché gli algoritmi per i problemi dei banditi assumono che una prova fallita abbia un costo reale in denaro.

Un'importante generalizzazione del processo dei banditi è il **superprocesso dei banditi** o **BSP** (*bandit superprocess*), in cui ogni braccio è un processo decisionale di Markov completo di per sé, e non un processo di ricompensa di Markov con una sola azione possibile. Tutte le altre proprietà rimangono identiche: le braccia sono indipendenti, soltanto un braccio (o un numero limitato) può essere utilizzato per volta, ed esiste un singolo fattore di sconto.

superprocesso
dei banditi

Tra gli esempi di BSP c'è la vita quotidiana, in cui si può eseguire un'attività per volta, anche se le attività che richiedono attenzione possono essere molte; il project management con progetti multipli; l'insegnamento a più allievi che necessitano di una guida individuale; e così via. Normalmente per questi processi si utilizza il termine **multitasking**, talmente diffuso che ormai non lo si nota: quando formulano un problema decisionale nel mondo reale, gli analisti raramente chiedono se il cliente ha anche altri problemi non correlati.

multitasking

Si potrebbe ragionare in questo modo: "Se ci sono n MDP disgiunti, è ovvio che una politica ottima complessiva possa essere costruita a partire dalle soluzioni ottime dei singoli MDP. Data la sua politica ottima π_i , ogni MDP diventa un processo di ricompensa di Markov in cui c'è una sola azione $\pi_i(s)$ in ogni stato s . Abbiamo così ridotto il superprocesso dei banditi con n braccia a un processo dei banditi con n braccia". Per esempio, se un immobiliarista ha una sola squadra di operai edili e diversi centri commerciali da costruire, sembra semplicemente buon senso l'ideazione di un piano di costruzione ottimo per ogni centro commerciale e poi la risoluzione del problema dei banditi per decidere dove mandare la squadra di operai ogni giorno.

Questo approccio appare plausibile, ma è sbagliato. In effetti, la politica ottima a livello globale per un BSP può includere azioni che sono localmente subottime dal punto di vista dell'MDP in cui sono effettuate. Il motivo è che la disponibilità di altri MDP in cui agire cambia l'equilibrio tra ricompense di breve e di lungo termine in un MDP componente, infatti tende a generare un comportamento più greedy in ogni MDP (cercando ricompense

nel breve termine) dato che puntare a una ricompensa nel lungo termine in un unico MDP causerebbe un ritardo delle ricompense in tutti gli altri MDP.

Per esempio, supponiamo che il piano di costruzione localmente ottimo per un centro commerciale preveda che il primo negozio sia disponibile all'affitto alla settimana 15, mentre un piano subottimo costa di più, ma consente di rendere disponibile il primo negozio alla settimana 5. Se ci sono quattro centri commerciali da costruire, potrebbe essere conveniente usare il piano localmente subottimo in ognuno di essi, in modo da poter cominciare a incassare gli affitti alle settimane 5, 10, 15 e 20, anziché attendere le settimane 15, 30, 45 e 60. In altre parole, quello che è un ritardo di 10 settimane per un singolo MDP diventa un ritardo di 40 settimane per il quarto MDP. In generale, le politiche globalmente e localmente ottime coincidono necessariamente soltanto quando il fattore di sconto è 1; in quel caso non ci sono costi associati al ritardo delle ricompense in un MDP.

A questo punto occorre pensare a come risolvere i BSP. Ovviamente la soluzione globalmente ottima per un BSP potrebbe essere calcolata trasformando il problema in un MDP globale sullo spazio degli stati del prodotto cartesiano, tuttavia il numero di stati sarebbe esponenziale nel numero di braccia del BSP, perciò la situazione si complicherebbe moltissimo.

Possiamo invece sfruttare il fatto che l'interazione tra le braccia è per natura molto ridotta, infatti tale interazione è ridotta fatto che l'agente ha una capacità limitata di utilizzare contemporaneamente le braccia. L'interazione può essere modellata in qualche misura dalla nozione di **costo-opportunità**: a quanta utilità si rinuncia, per passo temporale, non dedicando quel tempo a un altro braccio? Più alto è il costo-opportunità, maggiore è la necessità di generare presto ricompense con un dato braccio. In alcuni casi, una politica ottima per un dato braccio non è influenzata dal costo-opportunità (banalmente, è così in un processo di ricompensa di Markov perché c'è una sola politica). In quel caso è possibile applicare una politica ottima trasformando quel braccio in un processo di ricompensa di Markov.

Tale politica ottima, se esiste, è detta **politica dominante**. Aggiungendo azioni agli stati è sempre possibile creare una versione rilassata di un MDP (cfr. Paragrafo 3.6.2) in modo che abbia una politica dominante, che fornisce così un limite superiore al valore delle azioni su un braccio. Si può anche calcolare un limite inferiore risolvendo separatamente ogni braccio (il che potrebbe portare a una politica complessivamente subottima) e poi calcolando gli indici di Gittins. Se il limite inferiore per l'agire con un braccio è maggiore dei limiti superiori per tutte le altre azioni, allora il problema è risolto; altrimenti, si ha la garanzia che una combinazione di ricerca in avanti e ricalcolo dei limiti riuscirà a individuare una politica ottima per il BSP. Con questo approccio è possibile risolvere BSP relativamente grandi (con 10^{40} stati o più) in pochi secondi.

costo-opportunità

politica dominante

MDP parzialmente osservabile

17.4 MDP parzialmente osservabili

La descrizione dei processi decisionali di Markov, nel Paragrafo 17.1, presupponeva che l'ambiente fosse **completamente osservabile**. In questo caso l'agente conosce sempre lo stato in cui si trova. Questo, unito all'ipotesi di Markov sul modello di transizione, significa che la politica ottima dipende solo dallo stato corrente.

Quando l'ambiente è solo **parzialmente osservabile** la situazione è, potremmo dire, molto meno chiara. L'agente non sa necessariamente in che stato si trova, per cui non può eseguire l'azione $\pi(s)$ raccomandata per tale stato. Inoltre, l'utilità e l'azione ottima in uno stato s non dipendono solo da s , ma anche da *dalla conoscenza dell'agente* nel momento in cui si trova in tale stato. Per queste ragioni, gli **MDP parzialmente osservabili** (o **POMDP** – *partially observable MDP* – che si pronuncia “pom-di-pi”) sono considerati molto più difficili degli MDP ordinari. Non possiamo evitare di considerare i POMDP, comunque, perché lo stesso mondo reale ne fa parte.

17.4.1 Definizione di POMDP

Per capire i POMDP, per prima cosa occorre definirli in modo appropriato. Un POMDP ha gli stessi elementi di un MDP – il modello di transizione $P(s'|s, a)$, le azioni $A(s)$ e la funzione di ricompensa $R(s, a, s')$ – ma, come i problemi di ricerca con osservazioni parziali del Paragrafo 4.4, ha anche un modello sensoriale $P(e|s)$. Qui, come nel Capitolo 14, il modello sensoriale specifica la probabilità di percepire una evidenza e nello stato s .⁵ Per esempio, possiamo convertire il mondo 4×3 della Figura 17.1 in un POMDP aggiungendo un sensore rumoroso o parziale invece di assumere che l’agente conosca con esattezza la propria posizione. Potremmo utilizzare il sensore rumoroso a quattro bit del Paragrafo 14.3.2, che rileva la presenza o l’assenza di un muro in ogni direzione con accuratezza $1 - \varepsilon$.

Come per gli MDP, possiamo ottenere rappresentazioni compatte di POMDP di grandi dimensioni utilizzando reti di decisione dinamiche (cfr. Paragrafo 17.1.4). Aggiungiamo variabili sensori \mathbf{E}_t , ipotizzando che le variabili di stato \mathbf{X}_t non siano direttamente osservabili. Il modello sensoriale di un POMDP, quindi, è dato da $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$. Per esempio potremmo aggiungere variabili sensori alla DDN della Figura 17.4, come $MisuraBatteria_t$ per stimare la carica attuale di $Batteria_t$, e $Tachimetro_t$ per stimare il modulo del vettore di velocità \mathbf{X}_t . Un sensore sonar $Muri_t$ potrebbe fornire stime delle distanze dal muro più vicino in ognuna delle quattro direzioni cardinali riferite all’orientamento attuale del robot; questi valori dipendono dalla posizione corrente e dall’orientamento \mathbf{X}_t .

Nei Capitoli 4 e 11 abbiamo studiato i problemi di pianificazione in ambienti non deterministici e parzialmente osservabili e abbiamo identificato lo **stato-credenza** – l’insieme degli stati reali in cui l’agente potrebbe trovarsi – come concetto chiave per descrivere i problemi e calcolare le soluzioni. Nei POMDP lo stato-credenza b diventa una *distribuzione di probabilità* su tutti gli stati possibili, come nel Capitolo 14. Per esempio, lo stato-credenza iniziale per il POMDP 4×3 potrebbe essere la distribuzione uniforme sui nove stati non terminali combinata con zeri per gli stati terminali, cioè $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$.

Utilizziamo la notazione $b(s)$ per indicare la probabilità assegnata allo stato reale s dallo stato-credenza b . L’agente può calcolare il suo stato-credenza corrente come la distribuzione di probabilità condizionata su tutti gli stati reali, data la sequenza di percezioni e azioni passate. Questo è sostanzialmente il compito del **filtraggio** descritto nel Capitolo 14. L’equazione ricorsiva base del filtraggio (14.5) mostra come calcolare il nuovo stato-credenza partendo da quello precedente e dalle nuove evidenze. Nel caso dei POMDP dobbiamo anche considerare un’azione, ma il risultato è essenzialmente lo stesso. Se b è lo stato-credenza precedente e l’agente esegue l’azione a percepisce l’evidenza e , il nuovo stato-credenza si ottiene calcolando la probabilità di trovarsi ora nello stato s' , per ogni stato s' , con la seguente formula:

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s, a) b(s),$$

dove α è la costante di normalizzazione che fa sì che la somma sia pari a 1. Per analogia con l’operatore di aggiornamento per il filtraggio (Paragrafo 14.2.1), possiamo scrivere la precedente equazione come:

$$b' = \alpha \text{FORWARD}(b, a, e). \quad (17.16)$$

Nel POMDP 4×3 , supponiamo che l’agente si muova a *Sinistra* e il suo sensore rilevi un muro adiacente; allora è probabile (anche se non garantito, perché sia il movimento che il

⁵ Il modello sensoriale può anche dipendere dall’azione e dallo stato risultato, ma questo cambiamento non è fondamentale.



sensore sono rumorosi) che l’agente ora si trovi in (3,1). L’Esercizio 17.POMDP vi chiede di calcolare i valori di probabilità esatti per il nuovo stato-credenza.

L’intuizione fondamentale necessaria per comprendere i POMDP è questa: *l’azione ottima dipende solo dallo stato-credenza corrente dell’agente*. In altre parole, una politica ottima può essere descritta specificando la corrispondenza $\pi^*(b)$ tra stati-credenza e azioni. La politica *non* dipende dallo stato *reale* in cui si trova l’agente: questa è una buona cosa, perché l’agente non lo conosce; tutto ciò che conosce è appunto lo stato-credenza. Di conseguenza, il ciclo di decisione di un agente POMDP può essere suddiviso nei tre passi seguenti:

1. dato lo stato-credenza corrente b , esegui l’azione $a = \pi^*(b)$;
2. osserva la percezione e ;
3. assegna $\text{FORWARD}(b, a, e)$ come stato-credenza corrente e ripeti.

Possiamo pensare che i POMDP richiedano di svolgere una ricerca nello spazio degli stati-credenza, come i metodi che abbiamo visto nel Capitolo 4 per i problemi senza sensori e contingenti. La differenza principale è che lo spazio degli stati-credenza di un POMDP è *continuo*, dato che ogni stato-credenza è una distribuzione di probabilità. Uno stato-credenza nel mondo 4×3 , per esempio, è un punto in uno spazio continuo a 11 dimensioni. Un’azione modifica lo stato-credenza, e non solo quello fisico, perché influisce sulla percezione ricevuta. Quindi, l’azione è valutata almeno in parte in base all’informazione acquisita di conseguenza dall’agente. I POMDP includono dunque il valore dell’informazione (cfr. Paragrafo 16.6) come uno dei componenti del problema decisionale.

Ora consideriamo più attentamente gli esiti delle azioni: in particolare, calcoliamo la probabilità che un agente che si trova nello stato-credenza b raggiunga lo stato-credenza b' dopo aver eseguito l’azione a . Se conoscessimo l’azione e e la *successiva percezione*, l’Equazione (17.16) ci fornirebbe un aggiornamento *deterministico* dello stato-credenza: $b' = \text{FORWARD}(b, a, o)$. Naturalmente, però, la percezione successiva non è ancora nota, per cui l’agente potrebbe arrivare in uno tra molti possibili stati-credenza b' a seconda della percezione effettivamente ricevuta. La probabilità di percepire e , avendo eseguito a partendo dallo stato-credenza b , si ottiene mediante una somma su tutti gli stati fisici s' raggiungibili dall’agente:

$$\begin{aligned} P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e|s')P(s'|a,b) \\ &= \sum_{s'} P(e|s')\sum_s P(s'|s,a)b(s). \end{aligned}$$

Indichiamo la probabilità di raggiungere b' da b , data l’azione a , con $P(b'|b, a)$. Questa probabilità si può calcolare come segue:

$$\begin{aligned} P(b'|b,a) &= \sum_e P(b'|e,a,b)P(e|a,b) \\ &= \sum_e P(b'|e,a,b)\sum_{s'} P(e|s')\sum_s P(s'|s,a)b(s), \end{aligned} \tag{17.17}$$

dove $P(b'|e, a, b)$ vale 1 se $b' = \text{FORWARD}(b, a, e)$ e 0 in caso contrario.

Si può dire che l’Equazione (17.17) definisce un modello di transizione nello spazio degli stati-credenza. Possiamo anche definire una funzione di ricompensa per transizioni tra stati-credenza, derivata dalla ricompensa attesa per le transizioni tra stati reali che potrebbero verificarsi. In questo caso utilizziamo la forma semplice $\rho(b, a)$ per indicare la ricompensa attesa se l’agente esegue l’azione a nello stato-credenza b :

$$\rho(b,a) = \sum_s b(s) \sum_{s'} P(s'|s,a)R(s,a,s').$$



Insieme, $P(b' | b, a)$ e $\rho(b, a)$ definiscono un MDP *osservabile* nello spazio degli stati-credenza. Inoltre, può essere dimostrato che una politica che è ottima per questo MDP, $\pi^*(b)$, lo è anche per il POMDP originale. In altre parole, *la soluzione di un POMDP nello spazio degli stati fisici può essere ridotta a quella di un MDP nel corrispondente spazio degli stati-credenza*. Ciò è meno sorprendente se ripensiamo al fatto che uno stato-credenza è, per definizione, sempre osservabile dall'agente.

17.5 Algoritmi per risolvere POMDP

Abbiamo mostrato come ridurre i POMDP a MDP, ma gli MDP ottenuti in questo modo hanno uno spazio degli stati continuo (e solitamente con molte dimensioni). Questo significa che dovremo riprogettare gli algoritmi di programmazione dinamica dei Paragrafo 17.2.1 e 17.2.2, che presuppongono uno spazio degli stati finito e un numero finito di azioni. Nel seguito descriviamo un algoritmo di iterazione dei valori progettato specificamente per gli POMDP, e poi un algoritmo decisionale online simile a quelli sviluppati per i giochi nel Capitolo 5.

17.5.1 Iterazione dei valori per POMDP

Nel Paragrafo 17.2.1 abbiamo descritto un algoritmo di iterazione dei valori che calcola un unico valore di utilità per ogni stato. Con un numero di stati-credenza infinito, serve un tocco di creatività in più. Consideriamo una politica ottima π^* e la sua applicazione in uno specifico stato-credenza b : la politica genera un'azione, e poi, per ogni percezione successiva, lo stato-credenza viene aggiornato e viene generata una nuova azione, e così via. Per questo specifico b , quindi, la politica è esattamente equivalente a un **piano condizionale** come definito nel Capitolo 4 per problemi non deterministici e con osservazioni parziali. Anziché pensare alle politiche, pensiamo ai piani condizionali e a come l'utilità attesa di eseguire un piano condizionale fissato vari al variare dello stato-credenza iniziale. Facciamo due osservazioni:

1. Sia $\alpha_p(s)$ l'utilità di eseguire un piano condizionale *fissato* p partendo dallo stato fisico s . Allora l'utilità attesa di eseguire p nello stato-credenza b è $\sum_s b(s)\alpha_p(s)$, o $b \cdot \alpha_p$ se pensiamo entrambi come vettori. Quindi, l'utilità attesa di un piano condizionale fissato varia *linearmente* con b , ovvero corrisponde a un iperpiano nello spazio degli stati-credenza.
2. In qualsiasi stato-credenza b dato, una politica ottima sceglierà di eseguire il piano condizionale con l'utilità attesa più alta, e l'utilità attesa di b in una politica ottima è l'utilità di quel piano condizionale: $U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p$. Se una politica ottima π^* sceglie di eseguire p partendo da b , allora è ragionevole attendersi che possa scegliere di eseguire p in stati-credenza molto vicini a b ; in effetti, se imponiamo un limite alla profondità dei piani condizionali, il numero di tali piani è finito e lo spazio continuo degli stati-credenza sarà generalmente suddiviso in *regioni*, ognuna delle quali corrisponde a un particolare piano condizionale ottimo in essa.

Da queste due osservazioni ricaviamo che la funzione di utilità $U(b)$ su stati-credenza, che restituisce il massimo di una collezione di iperpiani, sarà *lineare a tratti e convessa*.

Per mostrare questo, utilizziamo un semplice mondo con due stati, etichettati A e B , e due azioni *Stai* che rimane nello stesso stato con probabilità 0,9, e *Vai* che passa all'altro stato con probabilità 0,9. Le ricompense sono $R(\cdot, \cdot, A) = 0$ e $R(\cdot, \cdot, B) = 1$; quindi, ogni transizione che termina in A ha ricompensa zero e ogni transizione che termina in B ha ricompensa 1. Per ora ipotizziamo come fattore di sconto $\gamma = 1$. Il sensore riporta lo stato corretto con probabilità 0,6. Ovviamente l'agente dovrebbe eseguire *Stai* quando si trova nello stato B e *Vai* quando si trova nello stato A . Il problema è che non sa dove si trova!

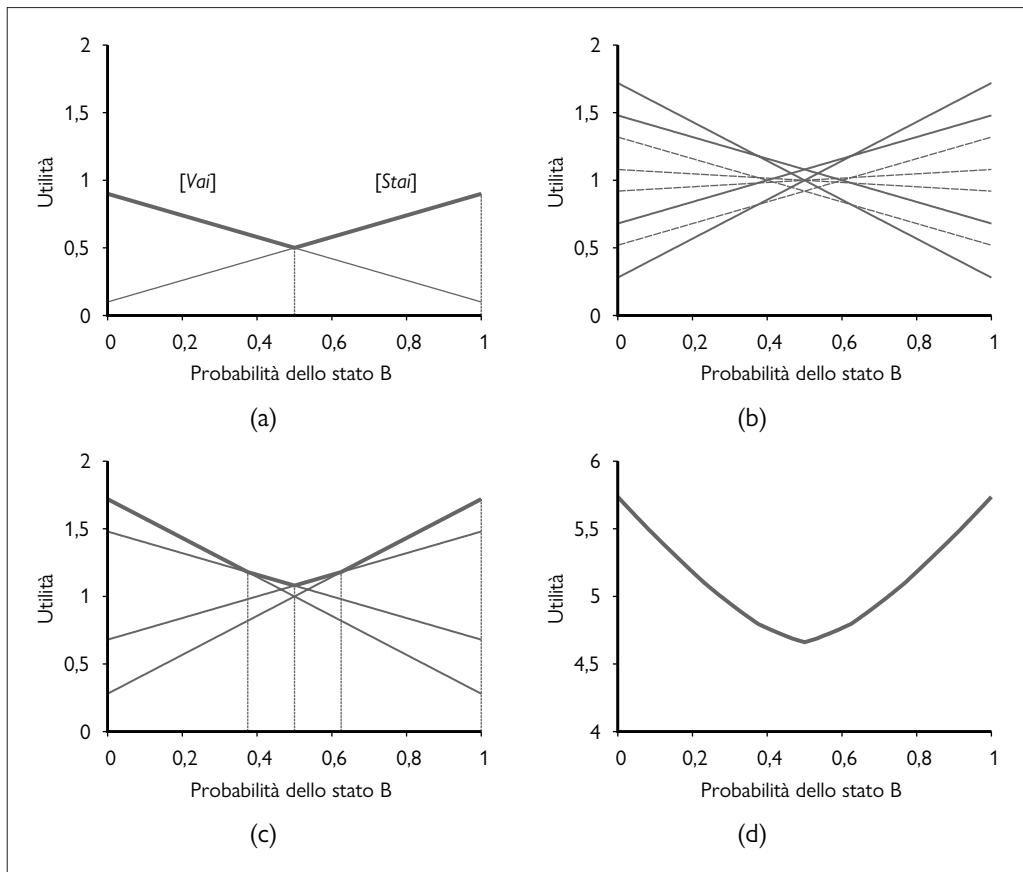


Figura 17.15 (a) Utilità di due piani a un solo passo in funzione dello stato-credenza iniziale $b(B)$ per il mondo a due stati, con la corrispondente funzione di utilità evidenziata con una linea spessa. (b) Utilità per 8 piani a due passi. (c) Utilità per 4 piani a due passi non dominati. (d) Funzione di utilità per piani ottimi a otto passi.

Il vantaggio di un mondo con due stati è che lo spazio degli stati-credenza può essere visualizzato in una sola dimensione, perché la somma delle due probabilità $b(A)$ e $b(B)$ è 1. Nella Figura 17.15(a), l'asse x rappresenta lo stato-credenza definito da $b(B)$, la probabilità di essere nello stato B . Ora consideriamo i piani a un passo $[Stai]$ e $[Vai]$, ognuno dei quali riceve la ricompensa per una transizione come segue:

$$\alpha_{[Stai]}(A) = 0,9R(A, Stai, A) + 0,1R(A, Stai, B) = 0,1$$

$$\alpha_{[Stai]}(B) = 0,1R(B, Stai, A) + 0,9R(B, Stai, B) = 0,9$$

$$\alpha_{[Vai]}(A) = 0,1R(A, Vai, A) + 0,9R(A, Vai, B) = 0,9$$

$$\alpha_{[Vai]}(B) = 0,9R(B, Vai, A) + 0,1R(B, Vai, B) = 0,1.$$

Gli iperpiani (in questo caso sono rette) per $b \cdot \alpha_{[Stai]}$ e $b \cdot \alpha_{[Vai]}$ sono mostrati nella Figura 17.15(a) e il loro massimo è evidenziato con uno spessore maggiore. La linea più spessa rappresenta quindi la funzione di utilità per il problema con orizzonte finito che consente una sola azione, e in ogni “tratto” della funzione di utilità lineare a tratti un’azione ottima è la prima azione del piano condizionale corrispondente. In questo caso, la politica ottima con un solo passo consiste nell'eseguire $Stai$ quando $b(B) > 0,5$ e Vai altrimenti.

Una volta ottenute le utilità $\alpha_p(s)$ per tutti i piani condizionali p di profondità 1 in ogni stato fisico s , possiamo calcolare le utilità per piani condizionali di profondità 2 considerando ogni possibile prima azione, ogni possibile percezione successiva, e poi ogni modo di scegliere un piano di profondità 1 da eseguire per ogni percezione:

```
[Stai; if Percezione = A then Stai else Vai]
[Stai; if Percezione = A then Stai else Vai]
[Vai; if Percezione = A then Stai else Stai]
...

```

Ci sono in tutto otto distinti piani di profondità 2, e le loro utilità sono mostrate nella Figura 17.15(b). Notate che quattro di questi piani, tracciati con linee tratteggiate, sono subottimi sull'intero spazio degli stati-credenza – diciamo che sono piani **dominati**, e non necessitano di ulteriore analisi. Ci sono quattro piani non dominati, ognuno dei quali è ottimo in una regione specifica, come mostrato nella Figura 17.15(c). Le regioni suddividono lo spazio degli stati-credenza.

piano dominato

Ora ripetiamo il processo per la profondità 3, e così via. In generale, sia p un piano condizionale di profondità d la cui azione iniziale è a e il cui sottopiano di profondità $(d - 1)$ per la percezione e è $p.e$; allora:

$$\alpha_p(s) = \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma \sum_e P(e | s') \alpha_{p.e}(s')]. \quad (17.18)$$

Questa ricorsione ci fornisce naturalmente un algoritmo di iterazione dei valori, riportato nella Figura 17.16. La struttura dell'algoritmo e l'analisi degli errori sono simili a quelli per l'algoritmo base di iterazione dei valori riportato nella Figura 17.6; la principale differenza è che, anziché calcolare un unico numero di utilità per ogni stato, POMDP-ITERAZIONE-VALORI mantiene una raccolta di piani non dominati con i loro iperpiani di utilità.

La complessità dell'algoritmo dipende principalmente dal numero di piani generati. Date $|A|$ azioni ed $|E|$ possibili osservazioni, ci sono $|A|^{O(|E|^{d-1})}$ piani di profondità d distinti. Per il mondo con due stati e $d = 8$, sono 2^{255} piani. L'eliminazione dei piani dominati è essenziale

```
function POMDP-ITERAZIONE-VALORI(pomdp,  $\varepsilon$ ) returns una funzione di utilità
  inputs: pomdp, un POMDP con stati  $S$ , azioni  $A(s)$ , modello di transizione  $P(s' | s, a)$ , modello sensoriale  $P(e | s)$ , ricompense  $R(s, a, s')$ , sconto  $\gamma$ 
          $\varepsilon$ , il massimo errore consentito nell'utilità di uno stato
  local variables:  $U$ ,  $U'$ , insiemi di piani  $p$  con vettori di utilità  $\alpha_p$ 

   $U' \leftarrow$  un insieme contenente tutti i piani a un solo passo  $[a]$ , con  $\alpha[a](s) = \sum_s P(s' | s, a) R(s, a, s')$ 
  repeat
     $U \leftarrow U'$ 
     $U' \leftarrow$  l'insieme di tutti i piani costituiti da un'azione  $e$ , per ogni possibile percezione successiva,
           un piano in  $U$  con vettori di utilità calcolati secondo l'Equazione (17.18)
     $U' \leftarrow$  RIMUOVI-PIANI-DOMINATI( $U'$ )
  until MAX-DIFFERENZA( $U$ ,  $U'$ )  $\leq \varepsilon(1 - \gamma)/\gamma$ 
  return  $U$ 
```

Figura 17.16 La struttura di alto livello dell'algoritmo di iterazione dei valori per POMDP. Il passo RIMUOVI-PIANI-DOMINATI e il test MAX-DIFFERENZA sono generalmente implementati come programmi lineari.

per ridurre questa crescita doppiamente esponenziale: il numero di piani non dominati con $d = 8$ è solo 144; la funzione di utilità per questi 144 piani è mostrata nella Figura 17.15(d).

Notate che gli stati-credenza intermedi hanno valore inferiore a quello dello stato A e dello stato B , perché negli stati intermedi l’agente non ha le informazioni necessarie per scegliere una buona azione. È per questo che le informazioni hanno un valore nel senso indicato nel Paragrafo 16.6 e le politiche ottime negli POMDP spesso includono azioni per ottenere informazioni.

Data una simile funzione di utilità, si può ricavare una politica eseguibile determinando quale iperpiano è ottimo in un qualsiasi stato-credenza dato b ed eseguendo la prima azione del piano corrispondente. Nella Figura 17.15(d), la politica ottima corrispondente è sempre la stessa per i piani di profondità 1: *Stai* quando $b(B) > 0,5$ e *Vai* altrimenti.

Nella pratica, l’algoritmo di iterazione dei valori della Figura 17.16 è inefficiente per problemi più grandi, già il POMDP 4×3 è troppo difficile, principalmente perché dati n piani condizionali non dominati al livello d , l’algoritmo costruisce $|A| \cdot n^{|E|}$ piani condizionali al livello $d + 1$ prima di eliminare quelli dominati. Con il sensore a quattro bit, $|E|$ è 16 e n può essere dell’ordine delle centinaia, perciò non c’è speranza di efficienza.

L’algoritmo è stato sviluppato negli anni 1970, e da allora sono stati compiuti vari progressi, tra cui forme più efficienti di iterazione dei valori e vari tipi di algoritmi di iterazione delle politiche. Alcuni di questi sono discussi nelle note bibliografiche al termine di questo capitolo. Per POMDP generali, comunque, trovare politiche ottime è molto difficile (PSPACE-difficile, in effetti, quindi difficilissimo). Nel paragrafo seguente descriviamo un metodo diverso, approssimato, per risolvere i POMDP, basato sulla ricerca in avanti.

17.5.2 Algoritmi online per POMDP

La struttura di base per un agente POMDP online è semplice: inizia con uno stato-credenza a priori; sceglie un’azione in base a un qualche processo di deliberazione basato sul suo stato-credenza corrente; dopo l’azione, riceve un’osservazione e aggiorna il suo stato-credenza usando un algoritmo di filtraggio; e poi il processo si ripete.

Una scelta ovvia per il processo di deliberazione è l’algoritmo expectimax descritto nel Paragrafo 17.2.4, ma con stati-credenza anziché stati fisici come nodi di decisione nell’albero. I nodi di casualità nell’albero POMDP hanno diramazioni etichettate con le osservazioni possibili, che conducono allo stato-credenza successivo, con le probabilità di transizione date dall’Equazione (17.17). Una porzione dell’albero expectimax per il POMDP 4×3 è mostrata nella Figura 17.17.

La complessità temporale di una ricerca esaustiva di profondità d è $O(|A|^d \cdot |E|^d)$, dove $|A|$ è il numero di azioni disponibili ed $|E|$ è il numero di possibili percezioni (notate che è molto minore del numero di possibili piani condizionali di profondità d generati dall’iterazione dei valori). Come nel caso osservabile, il campionamento nei nodi di casualità è un buon modo per ridurre il fattore di ramificazione senza perdere troppa accuratezza nella decisione finale. Quindi, la complessità di approssimare il processo decisionale online nei POMDP potrebbe non essere drasticamente peggiore rispetto agli MDP.

Per spazi degli stati molto grandi, il filtraggio esatto è impraticabile, perciò l’agente dovrà eseguire un algoritmo di filtraggio approssimato come il particle filtering (cfr. Paragrafo 14.5.3). Allora gli stati-credenza nell’albero expectimax diventano raccolte di particelle anziché distribuzioni di probabilità esatte. Per problemi con orizzonte lungo, potrebbe anche essere necessario eseguire le simulazioni usate nell’algoritmo UCT (Figura 5.11). La combinazione di particle filtering e UCT applicati a POMDP viene chiamata pianificazione di Monte Carlo con osservazioni parziali o **POMCP**. Con una rappresentazione del modello come rete DDN, l’algoritmo POMCP è applicabile, almeno in linea di principio, a POMDP molto grandi e realistici. I dettagli dell’algoritmo sono esaminati nell’Esercizio 17.POMC. La piani-

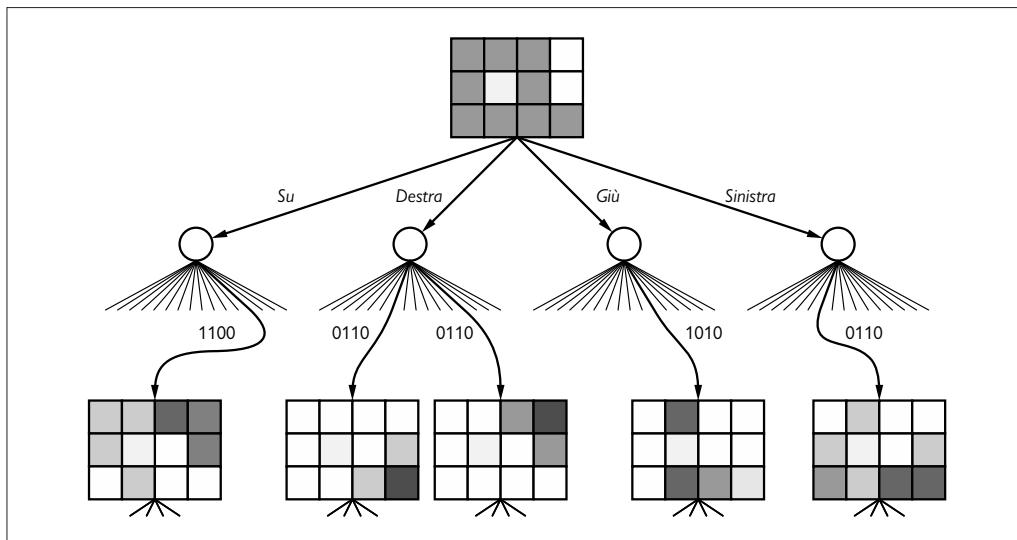


Figura 17.17
Parte di un albero expectimax per il POMDP 4×3 con uno stato-credenza iniziale uniforme. Gli stati-credenza sono illustrati con un livello di grigio proporzionale alla probabilità di trovarsi in ciascuna posizione.

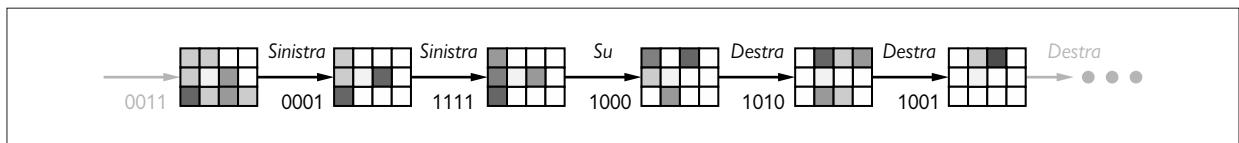


Figura 17.18 Una sequenza di percezioni, stati-credenza e azioni nel POMDP 4×3 con un errore di rilevamento dei muri $\epsilon = 0,2$. Notate che le prime mosse *Sinistra* sono sicure – è improbabile cadere $(4, 2)$ – e obbligano l’agente in un piccolo numero di posizioni possibili. Dopo l’azione *Su*, l’agente pensa di trovarsi probabilmente in $(3, 3)$, ma forse anche in $(1, 3)$. Fortunatamente, l’azione *Destra* è una buona idea in entrambi i casi, perciò l’agente esegue *Destra*, determina che è stato in $(1, 3)$ e ora si trova in $(2, 3)$, e poi continua andando a *Destra* e raggiunge l’obiettivo.

ficazione POMCP è in grado di generare un comportamento di qualità nel POMDP 4×3 . Un esempio breve (e in un certo senso fortunato) è mostrato nella Figura 17.18.

Gli agenti POMDP basati su reti di decisione dinamiche e processi di decisione online hanno diversi vantaggi rispetto ad altri agenti più semplici esaminati nei capitoli precedenti. In particolare, sono in grado di gestire ambienti stocastici parzialmente osservabili e di rivedere facilmente i loro “piani” per gestire evidenze inattese. Con modelli sensoriali appropriati, possono gestire anche il fallimento dei sensori e pianificare la raccolta di informazioni. In condizioni di pressione temporale e in ambienti complessi presentano un “degrado dolce” (*graceful degradation*) usando varie tecniche di approssimazione.

Che cosa manca allora? Il principale ostacolo all’impiego di questi agenti nel mondo reale è l’incapacità di generare comportamenti di successo su scale temporali lunghe. Simulazioni casuali o quasi casuali non hanno speranza di ottenere ricompense positive per compiti quali, per esempio, apparecchiare la tavola per cena, che potrebbe richiedere decine di milioni di azioni di controllo dei motori. Pare necessario ricorrere ad alcuni dei concetti di pianificazione gerarchica descritti nel Paragrafo 11.4. Nel momento in cui scriviamo non esistono ancora modi efficienti e soddisfacenti per applicare questi concetti in ambienti stocastici parzialmente osservabili.

17.6 Riepilogo

Questo capitolo ha mostrato come sfruttare la nostra conoscenza del mondo per prendere decisioni anche quando le loro conseguenze sono incerte, e le eventuali ricompense potrebbero non essere disponibili finché non sono state compiute diverse altre azioni. I punti principali sono i seguenti.

- I problemi di decisione sequenziali in ambienti stocastici, chiamati anche **processi decisionali di Markov** o MDP, sono definiti da un **modello di transizione** che specifica gli esiti probabilistici delle azioni e da una **funzione di ricompensa** che specifica la ricompensa associata a ogni stato.
- L'utilità di una sequenza di stati è pari alla somma di tutte le ricompense ricevute lungo la sequenza, con un possibile sconto dipendente dal tempo. La soluzione di un MDP è una **politica** che associa una decisione a ogni stato raggiungibile dall'agente. Una politica ottima massimizza l'utilità delle sequenze di stati attraversati durante la sua esecuzione.
- L'utilità di uno stato corrisponde alla somma delle ricompense attese con l'esecuzione di una politica ottima partendo da quello stato. L'algoritmo di **iterazione dei valori** risolve iterativamente le equazioni che collegano l'utilità di ogni stato a quelle dei suoi vicini.
- L'**iterazione delle politiche** alterna il calcolo delle utilità degli stati rispetto alla politica corrente con il miglioramento della politica in base alle utilità correnti.
- Gli MDP parzialmente osservabili, o POMDP, sono molto più difficili da risolvere degli MDP. È possibile convertire un POMDP in un MDP nello spazio continuo degli stati-credenza; sono stati progettati gli algoritmi di iterazione dei valori e di iterazione delle politiche. Il comportamento ottimo in un POMDP include la raccolta di informazioni per ridurre l'incertezza e prendere decisioni migliori.
- Per ambienti POMDP è possibile costruire agenti basati sulla teoria delle decisioni. Agenti siffatti utilizzano una rete di decisione dinamica per rappresentare il modello di transizione e il modello sensoriale, per aggiornare il proprio stato-credenza e per proiettare nel futuro possibili sequenze di azioni.

Torneremo su MDP e POMDP nel Capitolo 22 del Volume 2, quando studieremo i metodi di **apprendimento per rinforzo** che permettono a un agente di migliorare il proprio comportamento in base all'esperienza.

Note storiche e bibliografiche

Richard Bellman ha sviluppato i concetti alla base dell'approccio moderno verso i problemi di decisione sequenziali mentre lavorava presso RAND Corporation a partire dal 1949. Secondo la sua autobiografia (Bellman, 1984) coniò il termine “programmazione dinamica” per nascondere agli occhi del Segretario alla Difesa Charles Wilson, che temeva la ricerca, il fatto che il suo gruppo svolgesse ricerche di matematica (non può essere del tutto vero, perché il suo primo articolo in cui usava quel termine (Bellman, 1952) fu pubblicato prima che Wilson divenisse Segretario alla Difesa nel 1953). Il libro di Bellman, *Dynamic Programming* (1957), diede solide fondamenta al nuovo campo e introdusse l'algoritmo di iterazione dei valori.

Shapley (1953b) descrisse l'algoritmo di iterazione dei valori in modo indipendente da Bellman, ma i suoi risultati non trovarono ampio apprezzamento nella comunità della ricerca operativa, forse perché furono presentati nel contesto più generale dei giochi di Markov. Anche se le formulazioni originali includevano lo sconto, l'analisi in termini di preferenze stazionarie fu suggerita da Koopmans (1972). Il teorema dello shaping si deve a Ng *et al.* (1999).

La tesi di dottorato di Ron Howard (1960) introdusse l'iterazione delle politiche e l'idea della ricompensa media per risolvere i problemi a orizzonte infinito. Molti altri risultati furono presentati da Bellman e Dreyfus (1962). L'uso della mappatura delle contra-

zioni nell'analisi di algoritmi di programmazione dinamica si deve a Denardo (1967). L'iterazione delle politiche modificata si deve a van Nunen (1976) e Puterman e Shin (1978). L'iterazione delle politiche asincrona è stata analizzata da Williams e Baird (1993), che hanno anche dimostrato la limitatezza della perdita della politica nell'Equazione (17.13).

La famiglia generale degli algoritmi di **spazzamento con priorità** (*prioritized sweeping*) punta a velocizzare la convergenza verso politiche ottime mediante un ordinamento euristico dei calcoli per il valore e l'aggiornamento della politica (Moore e Atkeson, 1993; Andre *et al.*, 1998; Wingate e Seppi, 2005).

La formulazione della risoluzione di MDP come programma lineare si deve a de Ghellinck (1960), Manne (1960) e D'Épenoux (1963). Benché la programmazione lineare sia sempre stata considerata inferiore alla programmazione dinamica come metodo di risoluzione esatta per MDP, de Farias e Roy (2003) hanno mostrato che è possibile usare la programmazione lineare e una rappresentazione lineare della funzione di utilità per ottenere buone soluzioni approssimate di MDP molto grandi. Papadimitriou e Tsitsiklis (1987) e Littman *et al.* (1995) fornirono risultati generali sulla complessità computazione degli MDP. Yinyu Ye (2011) analizzò la relazione tra iterazione delle politiche e il metodo del simplesso per la programmazione lineare, dimostrando che, per γ fissato, il tempo di esecuzione dell'iterazione delle politiche è polinomiale nel numero di stati e azioni.

I fondamentali lavori di Sutton (1988) e Watkins (1989) sui metodi di apprendimento per rinforzo per la soluzione degli MDP hanno avuto un ruolo significativo per la loro diffusione nella comunità dell'IA (la precedente ricerca di Werbos (1977) conteneva molte idee simili, ma non ebbe un seguito così ampio). I ricercatori in IA hanno spinto gli MDP verso rappresentazioni più espansive che sono in grado di adattarsi a problemi molto più grandi rispetto alle tradizionali rappresentazioni atomiche basate su matrici di transizioni.

I concetti fondamentali per un'architettura di agenti basati sull'uso di reti di decisione dinamiche furono proposti da Dean e Kanazawa (1989a). Tatman e Shachter (1990) mostrarono come applicare algoritmi di programmazione dinamica a modelli DDN. Diversi autori studiarono la connessione tra MDP e problemi di pianificazione in IA, sviluppando forme probabilistiche della rappresentazione compatta STRIPS per modelli di transizione (Wellman, 1990b; Koenig, 1991). Il libro *Planning and Control* di Dean e Wellman (1991) esamina in modo molto approfondito tale connessione.

Lavori successivi sugli **MDP fattorizzati** (Boutilier *et al.*, 2000; Koller e Parr, 2000; Guestrin *et al.*, 2003b) usano rappresentazioni strutturate della funzione valore e del modello di transizione, con miglioramenti dimostrabili riguardo la complessità. Gli **MDP relativi** (Boutilier *et al.*, 2001; Guestrin *et al.*, 2003a) vanno un passo oltre, usando rappresentazioni strutturate per gestire domini con molti oggetti correlati. Gli MDP a universo aperto e i POMDP (Srivastava *et al.*, 2014b) supportano l'incertezza sull'esistenza e l'identità di oggetti e azioni.

Molti autori hanno sviluppato algoritmi online approssimati per processi decisionali basati su MDP, spesso prendendo a prestito esplicitamente approcci già esistenti in IA per problemi di ricerca e giochi in tempo reale (Werbos, 1992; Dean *et al.*, 1993; Tash e Russell, 1994). Il lavoro di Barto *et al.* (1995) sulla RTDP (*real-time dynamic programming*) fornì un quadro di riferimento generale per comprendere tali algoritmi e la loro connessione con l'apprendimento per rinforzo e la ricerca euristica. L'analisi di algoritmi expectimax a profondità limitata con campionamento nei nodi di casualità si deve a Kearns *et al.* (2002). L'algoritmo UCT descritto in questo capitolo si deve a Kocsis e Szepesvari (2006) e riprende lavori precedenti su simulazioni casuali per stimare i valori degli stati (Abramson, 1990; Brügmann, 1993; Chang *et al.*, 2005).

I problemi dei banditi furono introdotti da Thompson (1933) ma furono portati in primo piano dopo la seconda guerra mondiale con il lavoro di Herbert Robbins (1952). Bradt *et al.* (1956) dimostrarono i primi risultati sulle regole di arresto per banditi con un solo braccio, che portarono poi ai fondamentali risultati di John Gittins (Gittins e Jones, 1974; Gittins, 1989). Katehakis e Veinott (1987) suggerirono l'MDP di ripartenza come metodo per il calcolo degli indici di Gittins. Il testo di Berry e Fristedt (1985) tratta molte varianti del problema di base, mentre il chiarissimo testo online di Ferguson (2001) collega i problemi dei banditi con i problemi di arresto.

Lai e Robbins (1985) diedero l'avvio allo studio del rimpianto asintotico nelle politiche ottime per problemi dei banditi. L'euristica UCB fu introdotta e analizzata da Auer *et al.* (2002). I superprocessi dei banditi (BSP) furono studiati per primo da Nash (1973) ma sono rimasti poco conosciuti nell'IA. Hadfield-Menell e Russell (2015) hanno descritto un efficiente algoritmo branch-and-bound in grado di risolvere BSP relativamente grandi. I problemi di selezione furono introdotti da Bechhofer (1954). Hay *et al.* (2012) svilupparono una struttura formale per pro-

blemi di metaragionamento, mostrando che istanze semplici di tali problemi corrispondevano a problemi di selezione anziché a problemi dei banditi. Dimostrarono anche che il costo di calcolo atteso della strategia computazionale ottima non è mai più alto del guadagno atteso di qualità della decisione – anche se ci sono casi in cui la politica ottima può, con una certa probabilità, continuare a calcolare anche oltre il punto in cui è stato ottenuto ogni possibile guadagno.

L'osservazione che un MDP parzialmente osservabile può essere trasformato in un normale MDP su stati-credenza si deve ad Astrom (1965) e Aoki (1965). Il primo algoritmo completo per la soluzione esatta dei POMDP – in sostanza l'algoritmo di iterazione dei valori presentato in questo capitolo – fu proposto da Edward Sondik (1971) nella sua tesi di dottorato (un successivo articolo di Smallwood e Sondik (1973) contiene qualche errore, ma è più accessibile). Lovejoy (1991) fornì una panoramica sui primi venticinque anni della ricerca sugli POMDP, arrivando a conclusioni un po' pessimistiche sulla fattibilità di risolvere problemi di grandi dimensioni.

Nell'IA, il primo contributo significativo fu l'algoritmo Witness (Cassandra *et al.*, 1994; Kaelbling *et al.*, 1998), una versione migliorata dell'iterazione dei valori sui POMDP. Altri algoritmi seguirono poco dopo, tra cui un approccio dovuto a Hansen (1998) che costruisce incrementalmente una politica sotto forma di automa a stati finiti i cui stati definiscono i possibili stati-credenza dell'agente.

Lavori più recenti nell'IA si sono concentrati sui metodi di iterazione dei valori **puntuali** che, a ogni iterazione, generano piani condizionali e vettori α per un insieme finito di stati-credenza anziché per l'intero spazio degli stati-credenza. Lovejoy (1991) propose un algoritmo di quel tipo per una griglia fissata di punti, lo stesso approccio di Bonet (2002). Un importante articolo di Pineau *et al.* (2003) suggerì di generare punti raggiungibili simulando traiettorie in modo greedy; Spaan e Vlassis (2005) osservarono che occorre generare piani soltanto per un piccolo sottoinsieme scelto a caso di punti per migliorare rispetto ai piani delle iterazioni precedenti per tutti i punti dell'insieme. Shani *et al.* (2013) fornirono una panoramica su questi e altri sviluppi nel campo degli algoritmi puntuali, che portò a buone soluzioni per problemi con migliaia di stati. Poiché i POMDP sono PSPACE-dificili (Papadimitriou e Tsitsiklis, 1987), per compiere ulteriori progressi su metodi di soluzione offline potrebbe essere necessario ricorrere a vari tipi di struc-

ture nelle funzioni valore ottenute da una rappresentazione fattorizzata del modello.

L'approccio online per i POMDP – usando la ricerca in avanti per selezionare un'azione per lo stato-credenza corrente – fu esaminato per la prima volta da Satia e Lave (1973). L'uso del campionamento in nodi di casualità fu studiato analiticamente da Kearns *et al.* (2000) e da Ng e Jordan (2000). L'algoritmo POMCP si deve a Silver e Veness (2011).

Con lo sviluppo di algoritmi di approssimazione ragionevolmente efficaci per i POMDP, il loro uso come modelli per problemi del mondo reale è aumentato, soprattutto nei campi dell'istruzione (Rafferty *et al.*, 2016), dei sistemi di dialogo (Young *et al.*, 2013), della robotica (Hsiao *et al.*, 2007; Huynh e Roy, 2009) e delle auto a guida autonoma (Forbes *et al.*, 1995; Bai *et al.*, 2015). Un'importante applicazione su larga scala è l'Airborne Collision Avoidance System X (ACAS X), che evita le collisioni tra aerei e droni. Il sistema utilizza POMDP con reti neurali per l'approssimazione di funzioni. ACAS X migliora notevolmente la sicurezza rispetto al vecchio sistema TCAS, realizzato negli anni 1970 utilizzando la tecnologia dei sistemi esperti (Kochenderfer, 2015; Julian *et al.*, 2018).

I processi decisionali complessi sono stati studiati anche da economisti e psicologi, che hanno determinato che i processi decisionali non sono sempre razionali e non sempre operano esattamente come descritto in questo capitolo. Per esempio, la maggioranza delle persone preferisce ricevere 100 euro oggi alla garanzia di ricevere 200 euro tra due anni, ma le stesse persone preferiscono 200 euro tra otto anni a 100 euro tra sei anni. Questo risultato può essere interpretato nel senso che le persone non usano ricompense additive scontate esponenzialmente; forse usano **ricompense iperboliche** (la funzione iperbolica decade più bruscamente nel breve termine rispetto alla funzione di decadimento esponenziale). Queste e altre possibili interpretazioni sono state discusse da Rubinstein (2003).

I testi di Bertsekas (1987) e Puterman (1994) forniscono introduzioni rigorose ai problemi di decisione sequenziali e alla programmazione dinamica. Bertsekas e Tsitsiklis (1996) trattarono l'apprendimento per rinforzo. Sutton e Barto (2018) hanno trattato lo stesso tema ma in modo più accessibile. Sigaud e Buffet (2010), Mausam e Kolobov (2012) e Kochenderfer (2015) hanno trattato i processi decisionali sequenziali dal punto di vista dell'IA. Krishnamurthy (2016) ha fornito una panoramica completa sui POMDP.

Decisioni multiagente

- 18.1 Proprietà degli ambienti multiagente
- 18.2 Teoria dei giochi non cooperativi
- 18.3 Teoria dei giochi cooperativi
- 18.4 Decisioni collettive
- 18.5 Riepilogo
 - Note storiche e bibliografiche

In cui esaminiamo che cosa fare quando nell'ambiente operano più agenti.

18.1 Proprietà degli ambienti multiagente

Finora abbiamo assunto per lo più che percezione, pianificazione e azione siano compiute da un unico agente. Si tratta però di una assunzione estremamente semplificativa, che esclude molti scenari di IA reali. In questo capitolo, quindi, considereremo le questioni che sorgono quando un agente deve prendere decisioni in ambienti in cui operano più attori. Simili ambienti sono detti **sistemi multiagente** e gli agenti all'interno di un tale sistema affrontano un **problema di pianificazione multiagente**. Tuttavia, come vedremo, la precisa natura del problema di pianificazione multiagente, nonché le tecniche appropriate per risolverlo, dipendono dalle relazioni tra gli agenti presenti nell'ambiente.

18.1.1 Un solo decisore

La prima possibilità è che l'ambiente contenga più *attori* ma un unico *decisore*. In tal caso, il decisore sviluppa piani per gli altri agenti e comunica loro che cosa fare. L'ipotesi che gli agenti facciano semplicemente ciò che viene loro indicato è detta **ipotesi dell'agente benevolo**. Anche in questo scenario, però, i piani che coinvolgono più attori richiedono che essi *sincronizzino* le loro azioni. Gli attori *A* e *B* dovranno agire contemporaneamente per realizzare azioni congiunte (come cantare in duetto), in momenti diversi per azioni mutuamente esclusive (come ricaricare la propria batteria quando c'è solamente una presa), e sequenzialmente quando l'uno realizza le precondizioni per l'azione dell'altro (per esempio, *A* lava i piatti e *B* li asciuga).

pianificazione multiattuatore**pianificazione multibody****pianificazione decentralizzata****controparte****obiettivo comune****problema di coordinamento****teoria dei giochi****decisione strategica**

Un caso speciale è quello in cui un singolo decisore ha più attuatori che possono operare contemporaneamente. Ne è un esempio un essere umano in grado di camminare e parlare contemporaneamente. Un simile agente deve effettuare una **pianificazione multiattuatore** per gestire ciascuno degli attuatori e contemporaneamente le interazioni positive e negative tra di essi. Quando gli attuatori sono unità fisicamente separate, come nel caso di una flotta di robot fattorini in una fabbrica, la pianificazione multiattuatore diventa **pianificazione multibody** (“pianificazione multicorpo”).

Un problema multibody è ancora un problema “standard” con un unico agente, fintanto che le informazioni percettive raccolte da ciascun corpo possono essere aggregate, centralmente oppure in ciascun corpo, per formare una stima comune dello stato del mondo che poi informi l'esecuzione del piano generale; in tal caso, si può considerare che i diversi corpi agiscano come un corpo unico. Quando ciò non è possibile a causa di vincoli comunicativi, abbiamo ciò che a volte viene chiamato problema di **pianificazione decentralizzata** (decentralizzata); il termine è forse inappropriato, perché la fase di pianificazione è centralizzata mentre la fase di esecuzione è almeno parzialmente disaccoppiata. In questo caso, il sottopiano predisposto per ciascun corpo potrebbe richiedere esplicite azioni di comunicazione con gli altri corpi. Per esempio, dei robot da ricognizione che si muovono su un'area estesa potrebbero perdere spesso il contatto radio tra di loro e dovrebbero poi condividere le informazioni rilevate quando la comunicazione diviene nuovamente possibile.

18.1.2 Decisori multipli

La seconda possibilità è che anche gli altri attori presenti nell'ambiente siano decisori: ognuno ha le proprie preferenze e sceglie ed esegue il proprio piano. Li chiamiamo **controparti**. In questo caso possiamo distinguere due ulteriori possibilità.

- La prima è che, sebbene vi siano più decisori, essi persegua un **obiettivo comune**. È approssimativamente la situazione dei dipendenti di un'azienda, in cui differenti decisori persegono, auspicabilmente, i medesimi obiettivi nell'interesse dell'azienda. In questo scenario, il problema principale affrontato dai decisori è il **problema di coordinamento**: devono badare a remare tutti nella stessa direzione e a non intralciare accidentalmente i reciproci piani.
- La seconda possibilità è che ciascun decisore abbia le proprie preferenze individuali, che per seguirà al meglio delle proprie capacità. Può darsi che ci siano preferenze diametralmente opposte, come nel caso dei giochi a somma zero come gli scacchi (cfr. Capitolo 5). Ma la maggior parte delle situazioni multiagente è più complicata, con preferenze più complesse.

Quando sono presenti più decisori, ognuno dei quali persegue le proprie preferenze, un agente deve prendere in considerazione le preferenze degli altri agenti, oltre al fatto che gli altri agenti *a loro volta* prendono in considerazione le preferenze altrui, e così via. Entriamo così nel territorio della **teoria dei giochi**: la teoria delle decisioni strategiche. L'aspetto *strategico* del ragionamento, ovvero il fatto che ogni giocatore valuta il modo in cui gli altri giocatori potrebbero agire, è ciò che distingue la teoria dei giochi dalla teoria delle decisioni. Così come la teoria delle decisioni fornisce i fondamenti teorici per le decisioni nell'IA a un solo agente, la teoria dei giochi fornisce i fondamenti teorici per le decisioni nei sistemi multiagenti.

Nemmeno l'uso del termine “gioco” è ideale, in questo contesto: porta infatti a inferire che la teoria dei giochi riguardi principalmente attività ricreative, o scenari artificiali. Nulla di più lontano dalla realtà: la teoria dei giochi è la teoria delle **decisioni strategiche**; viene utilizzata per decisioni che riguardano per esempio le aste di diritti di esplorazione petrolifera e di frequenze radio, i procedimenti per bancarotta, lo sviluppo di prodotti e la scelta

dei prezzi, la difesa militare: situazioni che coinvolgono miliardi di dollari e molte vite. Nel campo dell'IA la teoria dei giochi può essere utilizzata principalmente in due modi.

- **Progettazione di agenti:** la teoria dei giochi può essere utilizzata da un agente per analizzare le decisioni possibili e calcolare l'utilità attesa di ciascuna (assumendo che gli altri agenti si comportino razionalmente, secondo la teoria dei giochi). In questo modo, con tecniche della teoria dei giochi si possono individuare la strategia migliore da adottare contro un giocatore razionale e il risultato atteso per ciascun giocatore. progettazione di agenti
- **Progettazione di meccanismi:** quando un ambiente è popolato da molti agenti, può darsi che sia possibile definire le regole di tale ambiente (cioè il gioco a cui gli agenti devono giocare) in modo tale che il bene collettivo degli agenti sia massimo quando ognuno adotta la soluzione che, in base alla teoria dei giochi, massimizza la sua utilità individuale. Per esempio, la teoria dei giochi può aiutare a progettare i protocolli per un insieme di router Internet in modo tale che ogni router sia incentivato a scegliere azioni che massimizzano il throughput globale. La progettazione di meccanismi può essere utilizzata anche per costruire sistemi multiagente intelligenti che risolvano problemi complessi in modo distribuito. progettazione di meccanismi

La teoria dei giochi fornisce una gamma di modelli differenti, a ognuno dei quali corrisponde una serie di assunzioni; è importante scegliere il modello giusto per ciascuno scenario. La distinzione più importante è quella tra giochi cooperativi e non.

- In un **gioco cooperativo** sono consentiti accordi vincolanti tra gli agenti, che rendono possibile una solida cooperazione. Nel mondo degli esseri umani, i contratti legali e le norme sociali contribuiscono a stabilire simili accordi vincolanti. Nel mondo dei programmi informatici, può esserci la possibilità di esaminare il codice sorgente per controllare che rispetti un certo accordo. Per analizzare questa situazione utilizzeremo la teoria dei giochi cooperativi. gioco cooperativo
- Se gli accordi vincolanti non sono possibili, abbiamo un **gioco non cooperativo**. Sebbene questo termine suggerisca che il gioco sia intrinsecamente competitivo e che la cooperazione sia impossibile, non è necessariamente così: "non cooperativo" significa semplicemente che non esiste un accordo centrale che vincoli tutti gli agenti e garantisca la cooperazione. Ma può darsi che gli agenti decidano autonomamente di cooperare perché è nel loro interesse farlo. Per analizzare questa situazione utilizzeremo la teoria dei giochi non cooperativi. gioco non cooperativo

In alcuni ambienti si combinano più dimensioni differenti. Per esempio, un'azienda che effettua consegne potrebbe adottare una pianificazione centralizzata per stabilire quotidianamente i percorsi dei propri automezzi e velivoli, lasciando tuttavia alcuni aspetti alle decisioni autonome degli autisti e dei piloti, i quali potranno reagire individualmente alle condizioni del traffico e a quelle meteorologiche. Gli obiettivi dell'azienda e quelli dei suoi dipendenti, inoltre, sono in certa misura resi convergenti dal pagamento di **incentivi** (salari e bonus); un chiaro segno del fatto che si tratta di un sistema multiagente. incentivo

18.1.3 Pianificazione multiagente

Per il momento tratteremo allo stesso modo gli scenari multiattuatore, multibody e multiagente, chiamandoli genericamente **scenari multiattore**; utilizziamo il termine generico **attore** per indicare attuatori, corpi (unità fisiche) e agenti. Obiettivo di questo paragrafo è trovare il modo per definire modelli di transizione, piani corretti e algoritmi di pianificazione efficienti per lo scenario multiattore. Un piano corretto è un piano che, quando eseguito dagli attori, raggiunge l'obiettivo (nello scenario multiagente, ovviamente, gli agenti potrebbero non essere d'accordo rispetto ad alcun particolare piano, ma sapranno quantomeno quali sono i piani che *funzionerebbero* se si *accordassero* per eseguirli).

**multiattore
attore**

concorrenza

Un problema chiave, quando si tratta di individuare un modello soddisfacente di azione multiattore, è che occorre gestire in qualche modo la spinosa questione della **concorrenza**, termine con cui intendiamo semplicemente che i piani dei diversi agenti potrebbero essere eseguiti in modo simultaneo. Se dobbiamo ragionare dell'esecuzione di piani multiattore, allora ci occorre dapprima un modello di piano multiattore che incorpori un adeguato modello di azione concorrente.

Inoltre, le azioni multiattore pongono un'intera serie di problemi che non si presentano nella pianificazione monoagente. In particolare, *gli agenti devono tenere conto di come le loro azioni interagiscono con quelle degli altri agenti*. Per esempio, un agente deve valutare se le azioni compiute dagli altri agenti possano disturbare le precondizioni delle sue azioni, se le risorse che utilizza nell'esecuzione della propria politica possano essere condivise e se sono suscettibili di essere consumate dagli altri agenti; se vi siano azioni reciprocamente esclusive; infine, un agente con atteggiamento collaborativo potrebbe considerare il modo in cui le sue azioni facilitano quelle degli altri.

Per rispondere a queste domande occorre un modello di azione concorrente entro cui sia possibile formularle in modo appropriato. I modelli di azione concorrente sono stati per decenni uno dei maggiori obiettivi della ricerca nella comunità scientifica dell'informatica, ma nessun modello ha prevalso affermandosi come definitivo e universalmente accettato. Non-dimeno, si sono ampiamente affermati i tre approcci seguenti.

esecuzione interleaved

Il primo approccio consiste nel considerare l'**esecuzione interleaved** (“interlacciata”) delle azioni dei diversi piani. Supponiamo per esempio di avere due agenti, *A* e *B*, con i seguenti piani:

$$\begin{aligned} A &: [a_1, a_2] \\ B &: [b_1, b_2]. \end{aligned}$$

L'idea alla base del modello dell'esecuzione interleaved è che l'unica certezza nell'esecuzione dei due piani sia il mantenimento dell'ordine delle azioni di ciascun piano. Se in aggiunta assumiamo che le azioni siano atomiche, allora esistono sei modi differenti per eseguire i due piani in concorrenza:

$$\begin{aligned} &[a_1, a_2, b_1, b_2] \\ &[b_1, b_2, a_1, a_2] \\ &[a_1, b_1, a_2, b_2] \\ &[b_1, a_1, b_2, a_2] \\ &[a_1, b_1, b_2, a_2] \\ &[b_1, a_1, a_2, b_2]. \end{aligned}$$



Per essere corretto nel modello dell'esecuzione interleaved, il piano *deve essere corretto rispetto a tutti i possibili interlacciamenti dei piani*. Il modello dell'esecuzione interleaved è stato ampiamente adottato nella comunità che si occupa della concorrenza, perché è un modello ragionevole del modo in cui più thread vengono eseguiti in alternanza su una singola CPU. Il modello tuttavia non copre il caso in cui due azioni avvengono contemporaneamente. Inoltre, il numero delle sequenze possibili cresce esponenzialmente con il numero di agenti e di azioni. Di conseguenza, verificare la correttezza di un piano, operazione semplice in uno scenario con un solo agente, con il modello dell'esecuzione alternata diventa difficoltoso dal punto di vista computazionale.

vera concorrenza

Il secondo approccio è quello della **vera concorrenza**, nel quale non tentiamo di creare un ordinamento serializzato completo delle azioni ma le lasciamo *parzialmente ordinate*: sappiamo che a_1 avverrà prima di a_2 , ma non siamo in grado di affermare nulla rispetto all'ordinamento di a_1 e b_1 , per esempio: potrebbero verificarsi l'una prima dell'altra, oppure contemporaneamente. Possiamo sempre “appiattire” un modello di ordinamento parziale

di piani concorrenti trasformandolo in un modello interleaved, ma così facendo perdiamo le informazioni sull'ordinamento parziale. Sebbene si possa sostenere che i modelli a ordinamento parziale siano più soddisfacenti di quelli interleaved per la rappresentazione teorica dell'azione concorrente, essi non sono stati adottati altrettanto diffusamente nella pratica.

Il terzo approccio consiste nell'assumere la perfetta **sincronizzazione**: c'è un orologio globale a cui tutti gli agenti hanno accesso, tutte le azioni richiedono lo stesso tempo e in ogni punto del piano congiunto le azioni sono simultanee. Così, le azioni di tutti gli agenti vengono compiute in modo sincrono, di pari passo (può accadere che alcuni agenti eseguano un'azione no-op quando attendono il completamento di altre azioni). In un contesto reale, l'esecuzione sincrona non è un modello molto completo di concorrenza, ma ha una semantica semplice, e per questo motivo è il modello con cui lavoreremo qui.

sincronizzazione

Cominciamo dal modello di transizione; per il caso deterministico a un solo agente, si tratta della funzione **RISULTATO**(s, a), che fornisce lo stato risultante dall'esecuzione dell'azione a quando l'ambiente è nello stato s . Nello scenario a un solo agente, potrebbero esistere b scelte differenti per l'azione; b può essere un numero piuttosto alto, specialmente per rappresentazioni del primo ordine con molti oggetti su cui agire, ma gli schemi di azione forniscono comunque una rappresentazione concisa.

Nello scenario multiattore con n attori, la singola azione a è sostituita da un'**azione congiunta** $\langle a_1, \dots, a_n \rangle$, dove a_i è l'azione compiuta dall'iesimo attore. Si presentano immediatamente due problemi: primo, dobbiamo descrivere il modello di transizione per b^n azioni congiunte; secondo, abbiamo un problema di pianificazione congiunta con un fattore di ramificazione b^n .

azione congiunta

Avendo collocato gli attori in un sistema multiattore con un elevato fattore di ramificazione, la pianificazione multiattore ha avuto come tema di ricerca principale il modo per *disaccoppiare* il più possibile gli attori, per far sì che (idealmente) la complessità del problema cresca linearmente con n invece che esponenzialmente con b^n .

Se gli attori non interagiscono gli uni con gli altri (per esempio n attori che giocano ognuno a un solitario) allora possiamo semplicemente risolvere n problemi distinti. Se gli attori sono **accoppiati debolmente** possiamo ottenere qualcosa che si avvicini a questo miglioramento esponenziale? È ovviamente una domanda cruciale in molti ambiti dell'IA. Abbiamo visto metodi di soluzione efficaci per sistemi debolmente accoppiati nel contesto dei CSP, dove grafi con vincoli "ad albero" fornivano metodi di soluzione efficienti (Paragrafo 6.5.2) e nel contesto dei database di pattern disgiunti (Paragrafo 3.6.3) e delle euristiche additive per la pianificazione (Paragrafo 11.3).

accoppiamento debole

L'approccio standard ai problemi debolmente accoppiati consiste nel fingere che i problemi siano completamente disaccoppiati e poi aggiustare le interazioni. Per il modello di transizione, ciò significa scrivere gli schemi di azione come se gli attori agissero in modo indipendente.

Vediamo come fare per una partita di tennis in doppio. Abbiamo due giocatori di tennis umani che formano una squadra di doppio e hanno l'obiettivo comune di vincere la partita contro la squadra avversaria. Supponiamo che, in un dato momento del gioco, la squadra abbia l'obiettivo di rimandare la palla nel campo avversario e che almeno uno dei due giocatori presidi la rete. La Figura 18.1 mostra le condizioni iniziali, l'obiettivo e gli schemi di azione per questo problema. È facile vedere che possiamo passare dalle condizioni iniziali all'obiettivo con un **piano congiunto** a due passi che specifichi ciò che ciascun giocatore deve fare: A deve spostarsi verso destra a fondo campo, mentre B deve solo rimanere fermo a rete:

piano congiunto

PIANO 1: $A : [Vai(A, FondocampoDestra), Colpisci(A, Palla)]$
 $B : [NoOp(B), NoOp(B)].$

Sorge però un problema se il piano prevede che i due agenti colpiscono la palla nello stesso momento. Nel mondo reale non funzionerebbe, ma lo schema d'azione per *Colpisci* afferma che la palla verrebbe giocata con successo. La difficoltà sta nel fatto che le precondizioni

Attori(A, B)

Init(Presso(A, FondocampoSinistra) \wedge Presso(B, ReteDestra)) \wedge

SiAvvicina(Palla, FondocampoDestra) \wedge Compagno(A, B) \wedge Compagno(B, A)

Obiettivo(Ribattuta(Ball) \wedge (Presso(x, ReteDestra) \vee Presso(x, ReteSinistra)))

Azione(Colpisci(attore, Palla),

PRECOND: SiAvvicina(Palla, pos) \wedge Presso(attore, pos) \wedge

EFFETTO: Ribattuta(Palla))

Azione(Vai(attore, a),

PRECOND: Presso(attore, pos) \wedge a \neq pos,

EFFETTO: Presso(attore, a) \wedge \neg Presso(attore, pos))

Figura 18.1 Il problema del tennis in doppio. Gli attori A e B giocano assieme e possono trovarsi in quattro diverse posizioni: *FondocampoSinistra*, *FondocampoDestra*, *ReteSinistra*, *ReteDestra*. La palla verrà ribattuta correttamente solo se un giocatore si trova nella posizione giusta. L'azione *NoOp* è fittizia, non ha effetto. Notate che ogni azione deve includere l'attore come argomento.

pongono dei vincoli riguardo allo stato in cui un'azione può essere eseguita con successo, ma non ne pongono riguardo alle azioni concorrenti che potrebbero comprometterne l'esito.

Si risolve questo problema introducendo negli schemi d'azione un nuovo elemento: un **vincolo di azione concorrente** che stabilisca quali azioni debbano o non debbano essere eseguite in concorrenza. Per esempio, l'azione *Colpisci* può essere descritta come segue:

Azione(Colpisci(attore, Palla),

CONCORRENTE: $\forall b \ b \neq attore \Rightarrow \neg Colpisci(b, Palla)$

PRECOND: Avvicina(Palla, pos) \wedge Presso(attore, pos)

EFFETTO: Ribattuta(Palla)).

In altre parole, l'azione *Colpisci* ha l'effetto dichiarato solo se nessun altro agente compie un'altra azione *Colpisci* nello stesso momento (nell'approccio SATPLAN, si ricorrerebbe a un **assioma di esclusione di azioni** parziale). Per alcune azioni, l'effetto desiderato viene ottenuto solo quando contemporaneamente ha luogo un'altra azione. Per esempio, per portare sul campo da tennis un frigorifero pieno di bibite sono necessari due agenti:

Azione(Porta(attore, frigor, qui, là),

CONCORRENTE: $\exists b \ b \neq attore \wedge Porta(b, frigor, qui, là)$

PRECOND: Presso(attore, qui) \wedge Presso(frigor, qui) \wedge Frigor(frigor)

EFFETTO: Presso(attore, là) \wedge Presso(frigor, là) \wedge \neg Presso(attore, qui) \wedge \neg Presso(frigor, qui)).

Con questi tipi di schema d'azione, ognuno degli algoritmi di pianificazione descritti nel Capitolo 11 può essere adattato, con lievi modifiche, per generare piani multiattore. Fintanto che il legame tra i sottopiani è debole, nel senso che i vincoli di concorrenza entrano in gioco solo raramente durante la ricerca del piano, ci si può aspettare che le euristiche derivate dalla pianificazione con agente singolo siano efficaci anche nel contesto multiattore.

18.1.4 Pianificazione con più agenti: cooperazione e coordinamento

Consideriamo ora uno scenario puramente multiagente nel quale ciascun agente crea il proprio piano. Per iniziare, ipotizziamo che gli obiettivi e le conoscenze siano condivisi. Si potrebbe pensare che ciò riconduca al caso multibody, in cui ciascun agente semplicemente

calcola la soluzione congiunta ed esegue la propria porzione di tale soluzione. Purtroppo, però, nella locuzione “la soluzione congiunta” il “la” è fuorviante. Ecco un secondo piano che a sua volta realizza l’obiettivo:

PIANO 2: $A : [Vai(A, ReteSinistra), NoOp(A)]$
 $B : [Vai(B, FondocampoDestra), Colpisci(B, Palla)].$

Se i due agenti riescono ad accordarsi sul piano 1 o sul piano 2, l’obiettivo verrà raggiunto. Ma se A sceglie il piano 2 mentre B sceglie il piano 1, allora nessuno ribatterà la palla. Gli agenti ne sono consapevoli, ma in che modo possono coordinarsi per far sì che vi sia accordo sul piano?

Una delle opzioni consiste nell’adottare una **convenzione** prima di intraprendere l’attività congiunta. Una convenzione è un vincolo sulla scelta tra piani congiunti. Per esempio, la convenzione “rimani dal tuo lato del campo” eliminerebbe il piano 1, ed entrambi i giocatori sceglierrebbero il piano 2. Su una strada, i conducenti dei veicoli hanno il problema di non scontrarsi tra loro, e nella maggior parte dei paesi tale problema viene (parzialmente) risolto adottando la convenzione “stai sul lato destro della strada”. L’alternativa, “stai sul lato sinistro”, funziona altrettanto bene a patto che tutti gli agenti dell’ambiente siano d’accordo. Considerazioni simili valgono per lo sviluppo del linguaggio umano, laddove l’importante non è stabilire quale lingua ciascun individuo debba parlare, ma il fatto che in una certa comunità tutti parlino la stessa lingua. Quando le convenzioni sono ampiamente diffuse, vengono dette **norme sociali**.

In assenza di una convenzione, gli agenti possono utilizzare la **comunicazione** per arrivare alla conoscenza comune di un piano congiunto praticabile. Per esempio, un tennista può urlare: “Mia!” oppure “Tua!” per indicare il piano congiunto che preferisce. La comunicazione non implica necessariamente uno scambio verbale. Per esempio, un giocatore può comunicare all’altro il piano congiunto desiderato semplicemente eseguendone la prima parte. Se l’agente A si avvicina alla rete, allora l’agente B è obbligato a retrocedere a fondo campo per colpire la palla, perché il piano 2 è l’unico piano congiunto che inizia con l’avvicinamento di A alla rete. Questo approccio al coordinamento, detto a volte **riconoscimento del piano**, funziona quando una singola azione (o una breve sequenza di azioni) da parte di un agente è sufficiente all’altro per individuare in modo non ambiguo un piano congiunto.

convenzione

norma sociale

**riconoscimento
del piano**

18.2 Teoria dei giochi non cooperativi

Introduciamo ora i concetti fondamentali e le tecniche analitiche della teoria dei giochi, la teoria che supporta le decisioni negli ambienti multiagente. La nostra trattazione inizia con la teoria dei giochi non cooperativi.

18.2.1 Giochi con una sola mossa: giochi in forma normale

Nel primo modello di gioco che esamineremo, tutti i giocatori agiscono simultaneamente e l’esito del gioco si basa sul profilo di azioni selezionato in questo modo (in realtà non è fondamentale che le azioni abbiano luogo nello stesso momento; ciò che conta è che nessun giocatore conosca le scelte degli altri). Questi giochi vengono detti **giochi in forma normale**. Un gioco in forma normale è definito da tre componenti:

**gioco in forma
normale**

- **Giocatori** o agenti che prendono decisioni. I giochi a due giocatori sono quelli che hanno ricevuto la maggiore attenzione, sebbene siano comuni anche i giochi a n giocatori con $n > 2$. Assegniamo ai giocatori nomi con l’iniziale maiuscola, come *Ali* e *Bo*, oppure *O* e *E*.
- **Azioni** che i giocatori possono scegliere. Diamo alle azioni nomi con l’iniziale minuscola, come *uno* oppure *testimoniare*. Non necessariamente i giocatori hanno a disposizione lo stesso insieme di azioni.

giocatori

funzione di payoff

- Una **funzione di payoff** che indichi l'utilità per ciascun giocatore di ciascuna combinazione delle azioni di tutti i giocatori. Per i giochi a due giocatori, la funzione di payoff di un giocatore può essere rappresentata da una matrice in cui vi sia una riga per ogni possibile azione di un giocatore e una colonna per ogni possibile azione dell'altro: la scelta di una riga e di una colonna individua una cella della matrice che contiene il payoff per il giocatore considerato. Nel caso a due giocatori, per convenzione si usa combinare le due matrici in una singola **matrice dei payoff**, nella quale ogni cella contiene i payoff di entrambi i giocatori.

matrice dei payoff

Per illustrare questi concetti, esaminiamo un gioco di esempio chiamato **morra a due dita**. In questo gioco i due giocatori, O e E , simultaneamente scelgono se mostrare un dito oppure due. Sia f il numero complessivo di dita mostrate: se f è dispari (*odd* in inglese), O riceve f euro da E ; se f è pari (*even* in inglese), E riceve f euro da O .¹ La matrice dei payoff della morra a due dita è la seguente:

	O: uno	O: due
E: uno	$E = +2, O = -2$	$E = -3, O = +3$
E: due	$E = -3, O = +3$	$E = +4, O = -4$

Le righe sono associate al giocatore E , le colonne al giocatore O . Quindi, per esempio, la cella in basso a destra mostra che quando sia O sia E scelgono l'azione *due*, il payoff è +4 per E e -4 per O .

Prima di analizzare la morra a due dita, vale la pena di considerare perché siano necessari i concetti della teoria dei giochi: perché non esaminare la situazione affrontata (poniamo) dal giocatore E mediante l'apparato della teoria delle decisioni e della massimizzazione dell'utilità che abbiamo utilizzato altrove in questo libro? Per comprendere perché serva altro, supponiamo che E stia provando a individuare la migliore azione da compiere. Le alternative sono *uno* e *due*. Se E sceglie *uno*, il payoff sarà +2 oppure -3. Il payoff che E riceverà *effettivamente*, però, dipende dalla scelta effettuata da O : E può decidere solamente in quale delle due righe si collocherà l'esito del gioco. Analogamente, O sceglie solo la colonna.

Per scegliere in modo ottimale tra queste possibilità, E deve prendere in considerazione il modo in cui O agirà come decisore razionale. Ma O , a sua volta, deve prendere in considerazione il fatto che E è un decisore razionale. Così, il processo decisionale negli scenari multiagente ha un carattere piuttosto differente rispetto a quello delle decisioni in scenari con un unico agente, perché i giocatori devono considerare il ragionamento altrui. Nella teoria dei giochi, il ruolo dei **concetti di soluzione** è quello di provare a rendere preciso questo tipo di ragionamento.

Il termine **strategia** viene utilizzato in teoria dei giochi per denotare ciò che in precedenza abbiamo definito *politica*. Una **strategia pura** è una politica deterministica; nei giochi a una sola mossa, una strategia pura è un'unica azione. Come vedremo in seguito, in molti giochi per l'agente è meglio adottare una **strategia mista**, cioè una politica randomizzata che seleziona le azioni sulla base di una distribuzione di probabilità. La strategia mista che sceglie l'azione a con probabilità p e l'azione b negli altri casi si indica con $[p: a; (1-p): b]$. Per esempio, una strategia mista per la morra a due dita potrebbe essere $[0,5: \text{uno}; 0,5: \text{due}]$. Un **profilo di strategie** è l'assegnazione di una strategia a ciascun giocatore. Dato il profilo di strategie, il **risultato** (*outcome*) del gioco è un valore numerico per ciascun giocatore. Se i giocatori adottano strategie miste, allora dobbiamo ricorrere all'utilità attesa.

concetto di soluzione**strategia**
strategia pura**strategia mista****profilo di strategie**

¹ La morra è una versione ricreativa di un **gioco di ispezione**. In questi giochi, un ispettore sceglie il giorno in cui eseguire un controllo (per esempio in un ristorante o in un impianto di produzione di armi batteriologiche) mentre il gestore del luogo ispezionato sceglie il giorno in cui nascondere tutte le irregolarità. L'ispettore vince se i giorni sono differenti, il gestore vince se i giorni coincidono.

In giochi come la morra, quindi, come si decide quale azione compiere? La teoria dei giochi fornisce una gamma di concetti di soluzione che tentano di definire l'azione razionale considerando le credenze di un agente circa le credenze dell'altro agente. Purtroppo, non esiste un unico concetto di soluzione perfetto: è problematico stabilire cosa significhi "razionale" quando ognuno dei due agenti sceglie solamente una parte del profilo di strategie che determina il risultato.

Introduciamo il primo concetto di soluzione mediante il gioco probabilmente più famoso nel corpus della teoria dei giochi: il **dilemma del prigioniero**. Questo gioco si basa su questo racconto: due presunti ladri, Ali e Bo, vengono arrestati perché colti vicino a un'abitazione appena svaligiata, e vengono interrogati separatamente. Gli inquirenti offrono a ognuno dei due un accordo: se testimonierà che il suo complice è il capo di un banda di scassinatori, sarà libero per avere collaborato, mentre il suo complice sconterà una pena di 10 anni in prigione. Invece, se entrambi testimoniano l'uno contro l'altro, saranno condannati entrambi a 5 anni di prigione. Ali e Bo sanno anche che, se si rifiuteranno entrambi di testimoniare, dovranno scontare solamente un anno di prigione ciascuno, per il reato più lieve di possesso di refurtiva. Ora Ali e Bo si trovano davanti al cosiddetto dilemma del prigionero: testimoniare oppure no? Essendo agenti razionali, Ali e Bo desiderano massimizzare la rispettiva utilità attesa, ovvero minimizzare il numero di anni di prigione. Ognuno dei due è indifferente rispetto al destino dell'altro. Il dilemma del prigionero è sintetizzato nella seguente matrice dei payoff:

	Ali: testimoniare	Ali: rifiutare
Bo: testimoniare	A = -5, B = -5	A = -10, B = 0
Bo: rifiutare	A = 0, B = -10	A = -1, B = -1

Dal punto di vista di Ali, la matrice dei payoff può essere analizzata nel modo seguente.

- Supponiamo che Bo scelga *testimoniare*. Allora Ali prende una condanna di 5 anni se fa altrettanto e di 10 anni se rifiuta, quindi in questo caso è meglio testimoniare.
- Viceversa, se Bo sceglie *rifiutare*, allora Ali sarà libera se testimonia e prenderà un anno se rifiuta, quindi testimoniare è meglio anche in questo caso.
- Quindi, indipendentemente da ciò che Bo sceglie di fare, per Ali è meglio testimoniare.

Ali ha scoperto che *testimoniare* è una **strategia dominante** in questo gioco. Diciamo che per il giocatore g la strategia s è **fortemente dominante** sulla strategia s' se per g il risultato di s è migliore di quello di s' per ogni scelta di strategia da parte dell'altro giocatore (o degli altri). La strategia s è **debolmente dominante** su s' se è migliore di quest'ultima in almeno un profilo di strategie e non è peggiore negli altri. Una strategia dominante è una strategia che domina su tutte le altre. Una assunzione comune nella teoria dei giochi è che un giocatore razionale sceglierà sempre una strategia dominante ed eviterà le strategie dominate. Essendo razionale (o non volendo essere considerata irrazionale), Ali sceglie la strategia dominante.

Non è difficile vedere che il ragionamento di Bo sarà identico: a sua volta concluderà che per lui *testimoniare* è una strategia dominante, e sceglierà di attuarla. La soluzione del gioco, secondo l'analisi delle strategie dominanti, sarà che entrambi sceglieranno *testimoniare* e di conseguenza entrambi verranno condannati a 5 anni di prigione.

In situazioni come questa, dove tutti i giocatori scelgono una strategia dominante, l'esito risultante è detto **equilibrio con strategie dominanti**. È un "equilibrio" perché nessun giocatore è incentivato a deviare modificando la propria scelta: per definizione, cambiando non migliorerebbe la propria condizione, e potrebbe peggiorarla. In questo senso, l'equilibrio con strategie dominanti è un concetto di soluzione molto forte.

Tornando al dilemma del prigionero, vediamo che il *dilemma* consiste nel fatto che il risultato dell'equilibrio con strategie dominanti, in cui entrambi i giocatori testimoniano, è peggiore per entrambi rispetto al risultato che otterrebbero se entrambi rifiutassero di testimoniare.

**dilemma
del prigioniero**

strategia dominante
**fortemente
dominante**
**debolmente
dominante**



**equilibrio con
strategie dominanti**

Il risultato di (*rifiutare, rifiutare*) sarebbe per ognuno dei due un anno di prigione, migliore per entrambi rispetto ai 5 anni a testa che derivano dall'equilibrio con strategie dominanti.

Esiste per Ali e Bo un modo per arrivare al risultato (*rifiutare, rifiutare*)? È certamente possibile per entrambi rifiutare di testimoniare, ma è difficile che degli agenti razionali possano compiere questa scelta, dato il modo in cui è strutturato il gioco. Ricordiamo che questo è un gioco non cooperativo: i due non possono parlare tra loro, quindi non possono stipulare un accordo vincolante di *rifiutare*.

È però possibile arrivare alla soluzione (*rifiutare, rifiutare*) se si modifica il gioco. Potremmo trasformarlo in un gioco cooperativo in cui agli agenti sia permesso stabilire un accordo vincolante. Oppure potremmo convertirlo in un **gioco ripetuto** in cui i giocatori sappiano che si incontreranno ancora; esamineremo questo caso più avanti. Oppure, i giocatori potrebbero avere delle convinzioni morali che favoriscono la cooperazione e la lealtà. Ma ciò significherebbe che i giocatori hanno funzioni di utilità differenti e, anche in questo caso, si tratterebbe di un gioco differente.

miglior risposta

La presenza di una strategia dominante facilita notevolmente il processo decisionale per un giocatore. Quando Ali capisce che testimoniare è una strategia dominante, non ha alcuna necessità di sforzarsi per prevedere ciò che farà Bo, perché sa che *indipendentemente da ciò che Bo farà*, testimoniare sarà per lei la **miglior risposta**. Tuttavia, la maggior parte dei giochi non ha né strategie dominanti né equilibri con strategie dominanti. È raro che una singola strategia sia la miglior risposta a tutte le possibili strategie della controparte.

equilibrio di Nash

Il prossimo concetto di soluzione che consideriamo è più debole dell'equilibrio con strategie dominanti, ma è applicabile a molte più situazioni; si chiama **equilibrio di Nash** in onore di John Forbes Nash, Jr. (1928–2015), che ne fece l'argomento della propria tesi di dottorato del 1950, lavoro per cui nel 1994 gli venne riconosciuto il premio Nobel.

Un profilo di strategie è un equilibrio di Nash se nessun giocatore può modificare unilateralmente la propria strategia in modo da ottenere un risultato migliore, assumendo che gli altri giocatori mantengano le rispettive strategie. Perciò, in un equilibrio di Nash tutti i giocatori giocano simultaneamente la miglior risposta rispetto alle scelte delle controparti. Un equilibrio di Nash rappresenta un punto stabile in un gioco: stabile nel senso che non esiste alcun incentivo razionale per alcun giocatore a cambiare scelta. Tuttavia, gli equilibri di Nash sono punti stabili *locali*: come vedremo, un gioco può contenere più equilibri di Nash.

Poiché la strategia dominante è la miglior risposta rispetto a *tutte* le strategie della controparte, ogni equilibrio con strategie dominanti deve essere anche un equilibrio di Nash (Esercizio 18.EQIB). Nel dilemma del prigioniero, quindi, esiste un unico equilibrio con strategie dominanti, che è anche l'unico equilibrio di Nash.

Il gioco dell'esempio seguente dimostra che a volte i giochi non hanno strategie dominanti e, inoltre, che alcuni giochi hanno più equilibri di Nash.

	Ali: I	Ali: r
Bo: t	A = 10, B = 10	A = 0, B = 0
Bo: b	A = 0, B = 0	A = 1, B = 1

È facile verificare che in questo gioco non ci sono strategie dominanti, per nessuno dei due giocatori, e che perciò non c'è alcun equilibrio con strategie dominanti. Tuttavia, i profili di strategie (t, I) e (b, r) sono entrambi equilibri di Nash. Ora, è chiaramente nell'interesse di entrambi puntare allo stesso equilibrio di Nash, (t, I) oppure (b, r), ma poiché siamo nell'ambito della teoria dei giochi *non cooperativi*, i giocatori devono effettuare le loro scelte in modo indipendente, senza alcuna conoscenza delle scelte degli altri e senza alcun modo per accordarsi con loro. È un esempio di **problema di coordinamento**: i giocatori vogliono coordinare le loro azioni globalmente, in modo da scegliere azioni che li conducano a un medesimo punto di equilibrio, ma devono farlo usando solamente decisioni locali.

Per risolvere i problemi di coordinamento sono stati proposti numerosi approcci. Un'idea è quella dei **punti focali**. In un gioco, un punto focale è un risultato che in qualche modo appare ai giocatori come il risultato “ovvio” rispetto al quale coordinare le loro scelte. Questa ovviamente non è una definizione precisa; il suo significato dipende dal gioco considerato. Nell'esempio precedente, però, esiste un ovvio punto focale: il risultato (t, l) darebbe a entrambi i giocatori un'utilità notevolmente maggiore di quella che otterrebbero se si coordinassero su (b, r) . Dal punto di vista della teoria dei giochi entrambi i risultati sono equilibri di Nash, ma sarebbe un giocatore irragionevole quello che si aspettasse un coordinamento su (b, r) .

Alcuni giochi non hanno equilibri di Nash con strategie pure, come illustrato dal gioco seguente, denominato **matching pennies** (“abbinamento delle monete”): Ali e Bo scelgono simultaneamente una faccia di una moneta, testa oppure croce: se scelgono allo stesso modo, allora Bo dà €1 ad Ali; se invece fanno scelte differenti, Ali dà €1 a Bo:

	Ali: testa	Ali: croce
Bo: testa	$A = 1, B = -1$	$A = -1, B = 1$
Bo: croce	$A = -1, B = 1$	$A = 1, B = -1$

Vi invitiamo a verificare che il gioco non contiene strategie dominanti e che nessuno degli esiti è un equilibrio di Nash con strategie pure: per ogni risultato, uno dei giocatori rimpiange la propria scelta e preferirebbe aver fatto una scelta diversa, data la scelta dell'altro.

Per trovare un equilibrio di Nash, il trucco è utilizzare strategie miste, ovvero consentire ai giocatori di randomizzare le scelte. Nash dimostrò che *con strategie miste, ogni gioco ha almeno un equilibrio di Nash*. Ciò spiega perché l'equilibrio di Nash è un concetto di soluzione così importante: altre soluzioni, come l'equilibrio con strategie dominanti, non necessariamente esistono in ogni gioco, ma se si cerca un equilibrio di Nash con strategie miste si ottiene sempre una soluzione.



Nel caso dei matching pennies, abbiamo un equilibrio di Nash con strategie miste se entrambi i giocatori scelgono *testa* e *croce* con uguale probabilità. Per verificare che questo risultato è in effetti un equilibrio di Nash, supponiamo che uno dei giocatori scelga un esito con una probabilità diversa da 0,5. L'altro giocatore ha allora la possibilità di avvantaggiarsi di ciò puntando tutto su una particolare strategia. Per esempio, supponiamo che Bo giochi *testa* con probabilità 0,6 (e quindi *croce* con probabilità 0,4). Allora per Ali la scelta migliore è giocare *testa* con certezza. È quindi facile constatare che il caso in cui Bo gioca *testa* con probabilità 0,6 non può fare parte di alcun equilibrio di Nash.

18.2.2 Benessere sociale

Nella teoria dei giochi, la prospettiva principale è quella dei giocatori all'interno del gioco, che tentano di ottenere i migliori risultati possibili per sé stessi. Tuttavia, a volte è istruttivo adottare una prospettiva differente. Supponiamo di essere un'entità benevola e onnisciente che osserva il gioco dall'alto, e di poter scegliere il risultato. Essendo benevoli, vorremo scegliere il miglior risultato complessivo, quello migliore per la *società nel suo complesso*, per così dire. Come scegliere? Quali criteri applicare? È la situazione in cui entra in gioco il concetto di **benessere sociale**.

benessere sociale

Probabilmente, il criterio più importante e meno controverso rispetto al benessere sociale è quello di evitare i risultati che determinano uno *spreco* di utilità. Questo requisito è colto dal concetto di **Pareto ottimalità**, così chiamata in riferimento all'economista italiano Vilfredo Pareto (1848–1923). Un risultato è ottimo (o efficiente) secondo Pareto se non esistono altri risultati che migliorino la condizione di un giocatore senza peggiorare quella di un altro. Scegliendo un risultato che non è Pareto-ottimo si spreca utilità, nel senso che sarebbe stato possibile incrementare l'utilità per almeno uno degli agenti, senza ridurre quella degli altri.

Pareto ottimalità

matching pennies

punto focale

benessere sociale utilitaristico

Il **benessere sociale utilitaristico** è una misura della bontà di un risultato in termini aggregati. Il benessere sociale utilitaristico di un risultato è semplicemente la somma delle utilità che quel risultato assegna ai giocatori. Il concetto di benessere sociale utilitaristico pone due difficoltà. La prima è che esso considera la somma ma non la *distribuzione* delle utilità tra i giocatori, e può quindi condurre a una distribuzione molto iniqua, se accade che tale distribuzione massimizzi la somma. La seconda difficoltà è che il concetto presuppone una *misura comune* per l'utilità. Molti economisti sostengono che sia impossibile individuarla perché l'utilità (a differenza del denaro) è una quantità soggettiva. Se dovessimo decidere come distribuire dei biscotti, dovremmo darli tutti a un mostro utilitaristico che affermasse: “Amo i biscotti mille volte più di chiunque altro”? Ciò massimizzerebbe l'utilità complessiva autodichiarata, ma non appare equo.

benessere sociale equalitario

La questione di come l'utilità è distribuita tra i giocatori è l'oggetto della ricerca sul **benessere sociale equalitario**. Per esempio, una delle proposte è che si debba massimizzare l'utilità attesa del membro della società che si trova nella situazione peggiore: un approccio maximin. Esistono anche altri criteri, tra cui il **coefficiente di Gini**, che coglie il grado di uniformità con cui l'utilità è distribuita tra i giocatori. Le principali difficoltà di queste proposte sono che esse a volte sacrificano un'ampia porzione del benessere totale per piccoli miglioramenti distributivi e che, così come l'utilitarismo puro, sono comunque esposte al rischio del mostro utilitaristico.

Applicando questi concetti al gioco del dilemma del prigioniero, presentato in precedenza, si spiega come mai esso sia definito dilemma. Ricordiamo che (*testimoniare, testimoniare*) è un equilibrio con strategie dominanti ed è l'unico equilibrio di Nash; tuttavia, è anche l'unico risultato che *non* è Pareto-ottimo. Il risultato (*rifiutare, rifiutare*) massimizza sia il benessere sociale utilitaristico, sia quello equalitario. Il dilemma sorge perciò dal fatto che una soluzione molto forte (l'equilibrio con strategie dominanti) conduce a un risultato che essenzialmente non soddisfa alcuno dei criteri che definiscono un risultato ragionevole dal punto di vista della “società”. Eppure non esiste, per i singoli giocatori, un modo chiaro per giungere a una soluzione migliore.

Calcolare gli equilibri

Consideriamo ora i problemi computazionali fondamentali associati ai concetti discussi. Consideriamo in primo luogo le strategie pure, nelle quali la randomizzazione non è consentita.

Se i giocatori hanno solamente un numero finito di possibili scelte, allora per trovare i punti di equilibrio si può effettuare una ricerca esaustiva: esaminare ogni possibile profilo di strategie e verificare se per almeno un giocatore sia vantaggioso deviare da quel profilo. Se la risposta è no, si tratta di un equilibrio di Nash con strategie pure. Le strategie dominanti e gli equilibri con strategie dominanti possono essere calcolati mediante algoritmi simili. Purtroppo, però, con n giocatori che dispongono ognuno di m possibili azioni, il numero dei possibili profili di strategie è m^n , troppo grande perché sia praticabile una ricerca esaustiva.

Un approccio alternativo, che funziona bene in alcuni giochi, è la **miglior risposta miope** (o **miglior risposta iterata**): si inizia scegliendo casualmente un profilo di strategie; se in quel profilo un giocatore sta facendo una scelta non ottima date le scelte degli altri, si sostituisce tale scelta con una ottima e poi si ripete il processo. Il processo convergerà se conduce a un profilo di strategie in cui ogni giocatore compie una scelta ottima date le scelte degli altri; in altre parole, un equilibrio di Nash. In alcuni giochi la miglior risposta miope non converge; è però garantito che lo sia in alcune importanti classi di giochi.

Calcolare gli equilibri con strategie miste è algoritmicamente più complicato. Per semplicità, ci concentreremo sui metodi per i giochi a somma zero e al termine del paragrafo parleremo brevemente della loro estensione ad altri giochi.

miglior risposta miope

Nel 1928, Von Neumann sviluppò un metodo per trovare la strategia mista *ottima* nei giochi **a somma zero** a due giocatori – giochi in cui la somma dei payoff è sempre zero (o una costante, come spiegato nel Paragrafo 5.1.1). La morra è chiaramente uno di questi giochi. Nei giochi a somma zero a due giocatori, è noto che i payoff sono uguali e opposti, quindi è sufficiente considerare i payoff di uno dei due, il massimizzatore (come nel Capitolo 5). Per la morra, scegiamo come massimizzatore il giocatore E associato ai risultati pari, e definiamo la matrice dei payoff come $U_E(e, o)$, ovvero il payoff ottenuto da E se E gioca e e O gioca o . Il metodo di Von Neumann è detto tecnica **maximin** e funziona nel modo seguente:

a somma zero

maximin

- Supponiamo di cambiare le regole e di stabilire che E sceglierà la propria strategia per primo e che la rivelerà a O ; poi O sceglierà la propria, conoscendo quella di E . Alla fine, valutiamo il payoff atteso del gioco in base alle strategie scelte. Ciò produce un gioco a turni al quale possiamo applicare l'algoritmo **minimax** standard del Capitolo 5. Supponiamo che ciò porti al risultato $U_{E,O}$. Dato che questo gioco favorisce palesemente O , la vera utilità U del gioco originale (dal punto di vista di E) è *almeno pari* a $U_{E,O}$. Per esempio, considerando solo le strategie pure, l'albero di gioco minimax ha un valore alla radice di -3 (Figura 18.2(a)), quindi sappiamo che $U \geq -3$.
- Supponiamo ora di modificare le regole in modo tale che sia O a rivelare per primo la propria strategia, seguito da E . Il valore minimax di questo gioco è allora $U_{O,E}$, e poiché questo gioco favorisce E sappiamo che U è *al massimo pari* a $U_{O,E}$. Con strategie pure, il valore è $+2$ (Figura 18.2(b)), quindi sappiamo che $U \leq +2$.

Combinando questi due argomenti, vediamo che la vera utilità U della soluzione del gioco originale deve soddisfare:

$$U_{E,O} \leq U \leq U_{O,E} \quad \text{ovvero, in questo caso,} \quad -3 \leq U \leq 2.$$

Per individuare il valore esatto di U , dobbiamo passare all'analisi con strategie miste. In primo luogo, osserviamo quanto segue: *dopo che il primo giocatore ha rivelato la propria strategia, per il secondo giocatore si tratta di scegliere una strategia pura*. La ragione è semplice: se il secondo giocatore gioca una strategia mista, $[p: uno; (1-p): due]$, la sua utilità attesa è una combinazione lineare $(p \cdot U_{uno} + (1-p) \cdot U_{due})$ delle utilità delle strategie pure, U_{uno} e U_{due} . Questa combinazione lineare non può mai essere migliore della migliore tra U_{uno} e U_{due} , quindi il secondo giocatore può semplicemente scegliere la migliore delle due.



Tenendo presente questa osservazione, si può pensare a degli alberi minimax che abbiano un numero infinito di rami alla radice, corrispondenti alle infinite strategie miste che il primo giocatore può scegliere. Ognuno di essi conduce a un nodo con due rami, corrispondenti alle strategie pure disponibili per il secondo giocatore. Possiamo rappresentare questi alberi infiniti in modo finito, collocando alla radice un'unica scelta “parametrizzata”, nel modo seguente.

- Se E sceglie per primo, la situazione è come nella Figura 18.2(c). E sceglie la strategia $[p: uno; (1-p): due]$ alla radice, poi O sceglie una strategia pura (e quindi una mossa) dato il valore di p . Se O sceglie *uno*, il payoff atteso (per E) è $2p - 3(1-p) = 5p - 3$; se O sceglie *due*, il payoff atteso è $-3p + 4(1-p) = 4 - 7p$. Possiamo rappresentare questi due payoff come rette in un grafico in cui p varia da 0 a 1 sull'asse x , come nella Figura 18.2(e). O , il minimizzatore, sceglierà sempre la retta che si trova più in basso, come indicato nella figura dal maggiore spessore dei segmenti. Quindi, il meglio che E possa fare alla radice è scegliere che p sia al punto di intersezione, ovvero

$$5p - 3 = 4 - 7p \quad \Rightarrow \quad p = 7/12.$$

L'utilità per E in questo punto è $U_{E,O} = -1/12$.

- Se O gioca per primo, la situazione è quella mostrata nella Figura 18.2(d). O sceglie la strategia $[q: uno; (1-q): due]$ alla radice e E sceglie la propria mossa dato il valore di q . I

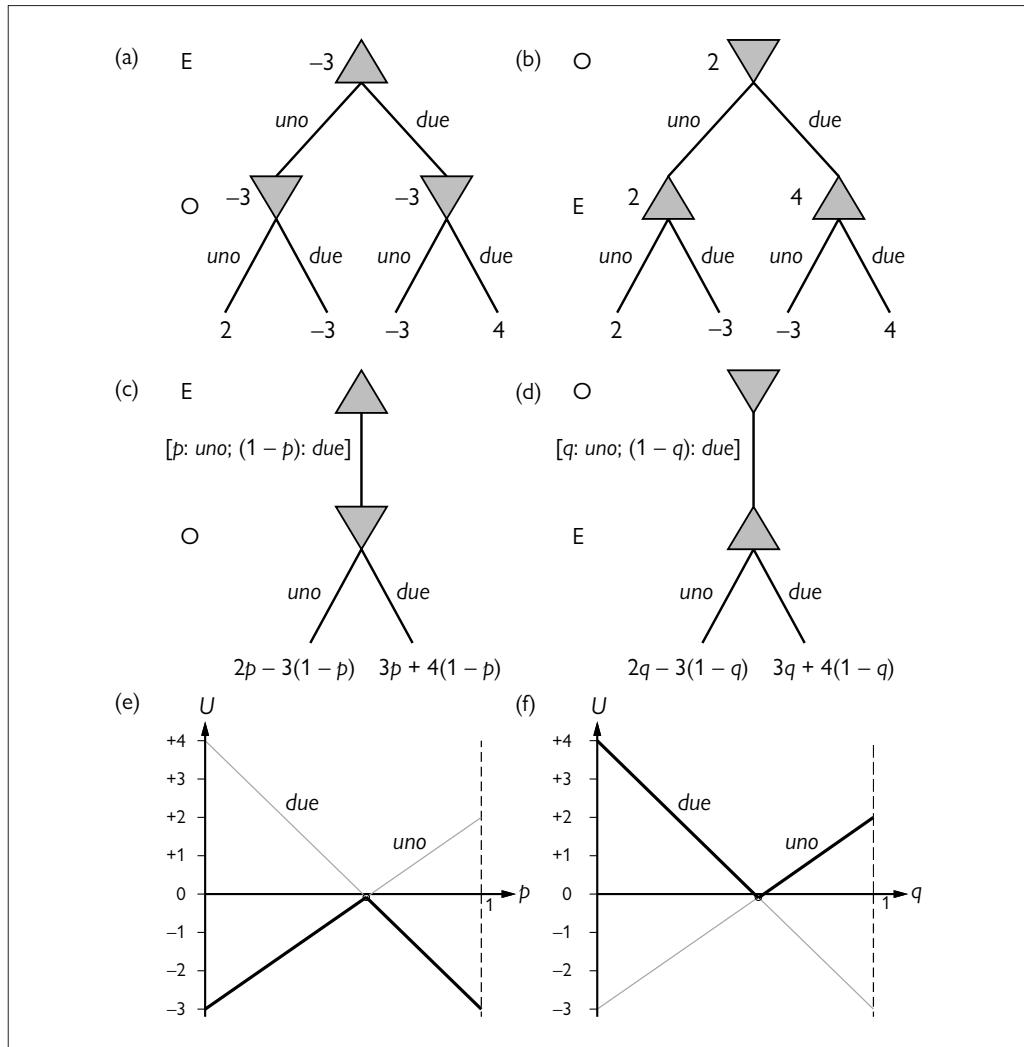


Figura 18.2 (a) e (b): alberi di gioco minimax per la morra a due dita, se si gioca a turno e con strategie pure. (c) e (d): alberi di gioco parametrizzati in cui il primo giocatore sceglie una strategia mista. I payoff dipendono dal parametro di probabilità (p o q) della strategia mista. (e) e (f): per ogni valore del parametro di probabilità, il secondo giocatore sceglierà l'azione “migliore” tra le due, perciò il valore della strategia mista del primo giocatore è indicato dalle linee più spesse. Il primo giocatore sceglierà come parametro di probabilità per la strategia mista il valore che corrisponde al punto di intersezione.

payoff sono $2q - 3(1 - q) = 5q - 3$ e $-3q + 4(1 - q) = 4 - 7q$.² Anche in questo caso, la Figura 18.2(f) mostra che il meglio che O possa fare è scegliere il punto di intersezione:

$$5q - 3 = 4 - 7q \quad \Rightarrow \quad q = 7/12.$$

L'utilità per E in questo punto è $U_{O,E} = -1/12$.

² Il fatto che queste equazioni siano uguali a quelle per p è una coincidenza dovuta a $UE(\text{uno}, \text{due}) = UE(\text{due}, \text{uno}) = -3$. Ciò spiega anche perché la strategia ottimale è la medesima per entrambi i giocatori.

Ora sappiamo che la vera utilità del gioco originale è compresa tra $-1/12$ e $-1/12$; ovvero, che è esattamente $-1/12$! (la conclusione è che in questo gioco è meglio essere *O* piuttosto che *E*). Inoltre, questa utilità si ottiene tramite la strategia mista $[7/12: uno; 5/12: due]$, che dovrebbe essere adottata da entrambi i giocatori. Questa strategia è detta **equilibrio maximin** del gioco ed è un equilibrio di Nash. Notate che tutte le strategie che compongono una strategia mista di equilibrio hanno la stessa utilità attesa. In questo caso, sia *uno* sia *due* hanno la stessa utilità attesa della strategia mista, $-1/12$.

Il nostro risultato per la morra a due dita è un esempio del risultato generale di Von Neumann: *quando sono ammesse le strategie miste, ogni gioco a somma zero tra due giocatori ha un equilibrio maximin*. Inoltre, ogni equilibrio di Nash in un gioco a somma zero è un equilibrio maximin per entrambi i giocatori. Un giocatore che adotta la strategia maximin ha due garanzie: primo, che non ci sia alcuna strategia migliore, se l'avversario gioca bene (potrebbero invece esserci strategie migliori a fronte di errori irrazionali dell'avversario); secondo, che la propria condizione non cambi nemmeno se la strategia viene rivelata all'avversario.



L'algoritmo generale per individuare gli equilibri maximin nei giochi a somma zero è più complicato di quanto suggeriscano le Figure 18.2(e) e (f). Quando le azioni possibili sono n , una strategia mista è un punto in uno spazio a n dimensioni e le rette diventano iperpiani. È anche possibile che alcune strategie pure del secondo giocatore siano dominate da altre, ovvero che non siano ottime rispetto ad *alcuna* strategia del primo giocatore. Una volta eliminate tutte queste strategie (processo che potrebbe essere necessario eseguire ripetutamente), la scelta ottima alla radice è il punto di intersezione più alto (o più basso) tra gli iperpiani rimanenti.

Individuare questa scelta è un problema di **programmazione lineare**: massimizzare una funzione obiettivo dati dei vincoli lineari. I problemi di questo tipo possono essere risolti con tecniche standard in tempo polinomiale nel numero di azioni (e nel numero di bit utilizzati per specificare la funzione di ricompensa, tecnicamente parlando).

Rimane la questione di cosa dovrebbe concretamente *fare* un agente razionale in una giocata singola della morra. L'agente razionale avrà determinato che $[7/12: uno; 5/12: due]$ è la strategia di equilibrio maximin e assumerà che anche un avversario razionale ne sia consapevole. L'agente potrebbe utilizzare un dado a 12 facce o un generatore di numeri casuali per scegliere casualmente secondo la strategia mista, nel qual caso il payoff atteso sarebbe $-1/12$ per *E*. Oppure potrebbe decidere di giocare semplicemente *uno* o *due*. In ogni caso, il payoff atteso per *E* è $-1/12$. Curiosamente, scegliere unilateralmente una particolare azione non ha effetti negativi sul proprio payoff atteso; tuttavia, rivelare all'altro agente di avere compiuto una tale decisione unilaterale *influisce* sul payoff atteso, perché l'avversario può adeguare la propria strategia corrispondentemente.

Trovare gli equilibri nei giochi a somma non zero è un po' più complicato. L'approccio generale prevede due passaggi: (1) enumerare tutti i possibili sottoinsiemi di azioni che potrebbero formare delle strategie miste. Per esempio, si provano prima tutti i profili di strategie in cui ogni giocatore utilizza una singola azione, poi quelli in cui ognuno utilizza una o due azioni, e così via. Questo processo è esponenziale nel numero di azioni, perciò è praticabile solo per giochi relativamente piccoli. (2) Per ogni profilo di strategie enumerato nel passaggio (1), verificare se si tratta di un equilibrio. Per farlo si risolve un sistema di equazioni e disequazioni simili a quelle del caso a somma zero. Con due giocatori le equazioni sono lineari e possono essere risolte con semplici tecniche di programmazione lineare, ma con tre o più giocatori sono non lineari e potenzialmente molto difficili da risolvere.

18.2.3 Giochi ripetuti

Finora abbiamo considerato solamente giochi che prevedono una sola mossa. Il tipo più semplice di gioco a più mosse è il **gioco ripetuto** (o **iterato**), in cui si gioca più volte un gioco

equilibrio maximin

gioco ripetuto

gioco base

a una sola mossa, chiamato **gioco base** (o *stage game*). Nei giochi ripetuti, la strategia specifica le azioni scelte dal giocatore in ciascuna ripetizione per ogni possibile successione di scelte precedenti.

Esaminiamo prima di tutto il caso in cui in cui il gioco base si ripete un numero di volte stabilito, finito e mutualmente noto a tutti; sono tutte condizioni necessarie per la validità dell'analisi che segue. Supponiamo che Ali e Bo giochino una versione ripetuta del dilemma del prigioniero e che entrambi sappiano di doverne giocare esattamente 100 ripetizioni. A ogni turno gli verrà chiesto se intendono *testimoniare* o *rifiutare*, e riceveranno un payoff per quel turno secondo le regole del dilemma del prigioniero già viste in precedenza.

Alla fine, si determinano i due payoff complessivi sommando tutti i payoff che ciascun giocatore ha raccolto nelle 100 ripetizioni. Quali strategie dovrebbero scegliere Ali e Bo in questo gioco? Ragioniamo come segue: entrambi sanno che la centesima ripetizione non sarà un gioco ripetuto, cioè il suo risultato non avrà effetto su ripetizioni successive; quindi, al centesimo turno si tratta in effetti di un singolo gioco del dilemma del prigioniero.

Come sappiamo, il risultato della centesima ripetizione sarà (*testimoniare*, *testimoniare*), la strategia dominante per entrambi i giocatori. Ma una volta determinata la centesima ripetizione, allora la novantanovesima non avrà alcun effetto su quelle successive, quindi a sua volta avrà come esito (*testimoniare*, *testimoniare*). Per induzione, entrambi i giocatori sceglieranno *testimoniare* a ogni turno, ottenendo un totale di 500 anni di prigione a testa. Questo tipo di ragionamento è noto come **induzione a ritroso** (*backward induction*) e ha un ruolo fondamentale nella teoria dei giochi.

Se però rinunciamo a una delle tre condizioni (che il numero di ripetizioni sia stabilito, finito e noto a tutti) il ragionamento induttivo non regge. Supponiamo che il gioco si ripeta un numero *infinito* di volte. Matematicamente, la strategia di un giocatore in un gioco ripetuto all'infinito è una funzione che associa ogni possibile andamento passato del gioco a una scelta del giocatore nella ripetizione considerata. Quindi una strategia guarda a ciò che è accaduto in precedenza nel gioco e decide quale scelta compiere per la ripetizione attuale. Ma non è possibile memorizzare una tabella infinita in un computer finito. Occorre un modello *finito* di strategie per giochi che verranno giocati un numero infinito di volte. Per questo motivo è consuetudine rappresentare le strategie per i giochi ripetuti infinitamente come macchine a stati finiti (MSF) con output.

La Figura 18.3 illustra una serie di strategie MSF per il dilemma del prigioniero ripetuto. Consideriamo la strategia **tit-for-tat** (ovvero “dente per dente”): ogni ovale rappresenta uno stato della macchina e al suo interno si trova la scelta che la strategia prevede quando la macchina è in quello stato. In uscita da ogni stato abbiamo un percorso per ogni possibile scelta della controparte: per trovare il successivo stato della macchina seguiamo la freccia corrispondente alla scelta effettuata dall'altro. Infine, uno degli stati è indicato da una freccia in ingresso, a indicare che si tratta dello stato iniziale. Con TIT-FOR-TAT, quindi, la macchina parte dallo stato *rifiutare*; se la controparte gioca *rifiutare*, la macchina rimane nel medesimo stato; se invece la controparte gioca *testimoniare*, la macchina passa allo stato *testimoniare*. Rimarrà nello stato *testimoniare* fino a quando la controparte giocherà *testimoniare*, ma se quest'ultima gioca *rifiutare*, tornerà allo stato *rifiutare*. Riepilogando, TIT-FOR-TAT inizia scegliendo *rifiutare* e poi si limita a copiare ciò che la controparte ha fatto nel turno precedente.

Le strategie FALCO e COLOMBA sono più semplici: FALCO gioca *testimoniare* a ogni tornata, mentre COLOMBA gioca sempre *rifiutare*. La strategia INFLESSIBILE è abbastanza simile a TIT-FOR-TAT ma con un'importante differenza: la prima volta che la controparte gioca *testimoniare*, la strategia si trasforma essenzialmente in FALCO: giocherà *testimoniare* all'infinito. Mentre TIT-FOR-TAT è indulgente, nel senso che ricambierebbe un successivo *rifiutare*, con INFLESSIBILE non c'è perdono. Giocare *testimoniare* un'unica volta è sufficiente per provocare una rappresaglia (giocare *testimoniare*) che prosegue all'infinito (sapreste definire cosa fa TAT-FOR-TIT?).

induzione a ritroso**tit-for-tat**

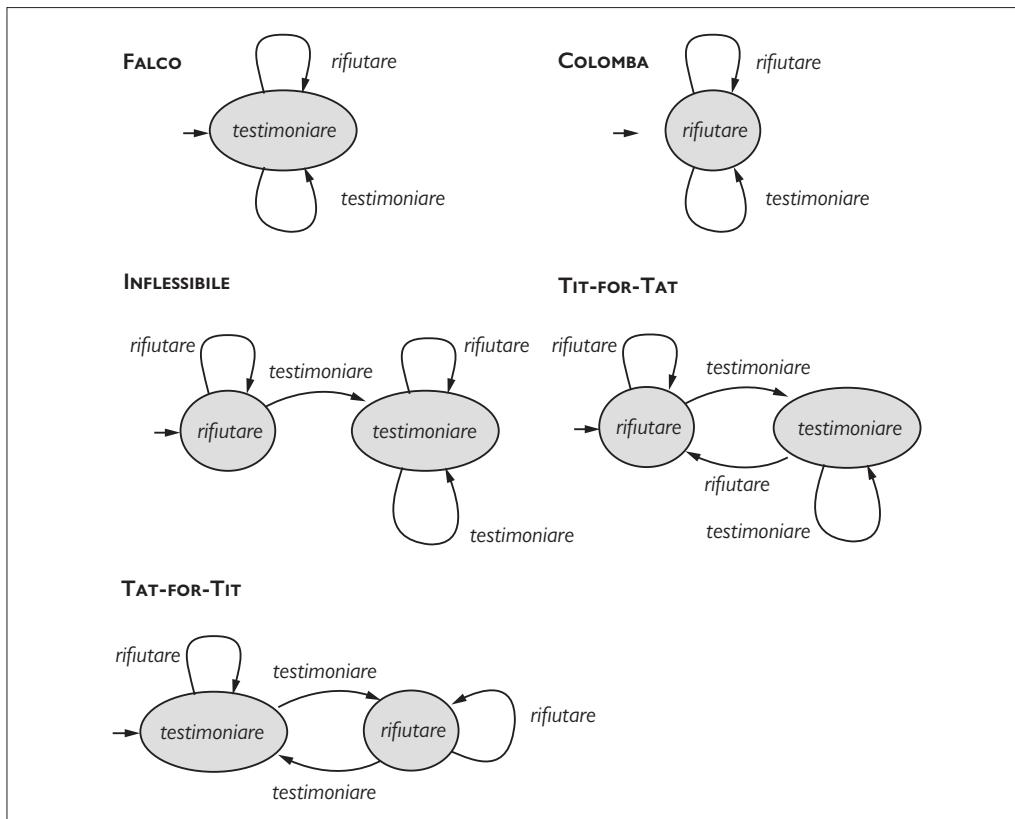


Figura 18.3
Alcune comuni strategie, dai nomi coloriti, rappresentate da macchine a stati finiti e applicabili al dilemma del prigioniero ripetuto all'infinito.

Un altro problema con i giochi che si ripetono all'infinito è quello di misurare l'utilità di una sequenza infinita di payoff. In questa sede ci concentreremo sull'approccio del **limite della media**, che essenzialmente consiste nel considerare la media delle utilità ricevute nella sequenza infinita. Con questo approccio, data una sequenza infinita di payoff (U_0, U_1, U_2, \dots), definiamo l'utilità della sequenza per il giocatore corrispondente come:

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^T U_t.$$

Non è garantito che questo valore converga per ogni sequenza arbitraria di utilità, ma è garantito che converga per le sequenze di utilità generate utilizzando strategie MSF. Per verificarlo, osserviamo che se alcune strategie MSF giocano l'una contro l'altra, *prima o poi torneranno a una configurazione che avevano già in precedenza, e da quel momento cominceranno a ripetersi*. Più precisamente, ogni sequenza di utilità generata da strategie MSF consisterà in una sequenza finita non ripetuta (eventualmente vuota), seguita da una sequenza finita non vuota che si ripete all'infinito. Per calcolare l'utilità media per un giocatore lungo tale sequenza infinita, è sufficiente calcolare la media della sequenza finita ripetuta.

Nel ragionamento seguente assumeremo che i giocatori di un gioco ripetuto all'infinito si limitino a scegliere una macchina a stati finiti che giochi per loro. Non poniamo alcun vincolo rispetto a queste macchine: possono essere grandi ed elaborate a piacimento dei giocatori. Quando tutti i giocatori hanno scelto una macchina a stati finiti che giochi per loro, possiamo calcolare i payoff per ciascun giocatore ricorrendo all'approccio del limite della media che abbiamo descritto. In questo modo, un gioco ripetuto all'infinito si riduce a un gioco in forma normale, seppur con infinite possibili strategie per ciascun giocatore.

Vediamo che cosa succede quando giochiamo il dilemma del prigioniero ripetuto all'infinito adottando alcune delle strategie della Figura 18.3. Supponiamo inizialmente che Ali e Bo scelgano entrambi COLOMBA.

0	1	2	3	4	5	...
Ali: COLOMBA	<i>rifiutare rifiutare rifiutare rifiutare rifiutare rifiutare ...</i>					utilità = -1
Bo: COLOMBA	<i>rifiutare rifiutare rifiutare rifiutare rifiutare rifiutare ...</i>					utilità = -1

Non è difficile vedere che questa coppia di strategie non costituisce un equilibrio di Nash: entrambi i giocatori si troverebbero in una condizione migliore passando alla scelta FALCO. Supponiamo allora che Ali passi a FALCO:

0	1	2	3	4	5	...
Ali: FALCO	<i>testimoniare testimoniare testimoniare testimoniare testimoniare testimoniare ...</i>					utilità = 0
Bo: COLOMBA	<i>rifiutare rifiutare rifiutare rifiutare rifiutare rifiutare ...</i>					utilità = -10

Questo è il peggior risultato possibile per Bo; e nemmeno questa coppia di strategie è un equilibrio di Nash. Anche per Bo sarebbe stato preferibile scegliere FALCO:

0	1	2	3	4	5	...
Ali: FALCO	<i>testimoniare testimoniare testimoniare testimoniare testimoniare testimoniare ...</i>					utilità = -5
Bo: FALCO	<i>testimoniare testimoniare testimoniare testimoniare testimoniare testimoniare ...</i>					utilità = -5

Questa coppia di strategie costituisce in effetti un equilibrio di Nash, ma non un equilibrio molto interessante: ci riporta più o meno al punto in cui eravamo con la versione singola del gioco, con i giocatori che scelgono di testimoniare l'uno contro l'altro. Ciò illustra una proprietà fondamentale dei giochi ripetuti all'infinito: *gli equilibri di Nash del gioco base persistono come equilibri in una versione del gioco con infinite ripetizioni*.

Ma non è ancora finita. Supponiamo che Bo passi a INFLESSIBILE:

0	1	2	3	4	5	...
Ali: FALCO	<i>testimoniare testimoniare testimoniare testimoniare testimoniare testimoniare ...</i>					utilità = -5
Bo: INFLESSIBILE	<i>rifiutare testimoniare testimoniare testimoniare testimoniare testimoniare ...</i>					utilità = -5

Bo non peggiora la propria situazione rispetto alla scelta di giocare FALCO: al primo turno, Ali gioca *testimoniare* mentre Bo sceglie *rifiutare*, ma ciò fa sì che Bo passi irreversibilmente all'opzione *testimoniare*: la perdita di utilità subita nella prima tornata scompare al limite. Nel complesso, i due giocatori ottengono la stessa utilità che avrebbero ottenuto giocando entrambi FALCO. Ma il punto è che queste strategie non costituiscono un equilibrio di Nash, perché questa volta Ali ha a disposizione una deviazione vantaggiosa: passare a INFLESSIBILE. Se entrambi scegliersero INFLESSIBILE, succederebbe questo:

0	1	2	3	4	5	...
Ali: INFLESSIBILE	<i>rifiutare rifiutare rifiutare rifiutare rifiutare rifiutare ...</i>					utilità = -1
Bo: INFLESSIBILE	<i>rifiutare rifiutare rifiutare rifiutare rifiutare rifiutare ...</i>					utilità = -1

I risultati e i payoff sono gli stessi che si avrebbero se entrambi avessero scelto COLOMBA, ma a differenza di quel caso, INFLESSIBILE contro INFLESSIBILE costituisce un equilibrio di Nash, e Ali e Bo sono in grado di raggiungere razionalmente un risultato che nella versione a un solo turno del gioco è impossibile.

Per verificare che queste strategie costituiscono un equilibrio di Nash, supponiamo per spirito di contraddizione che ciò non sia vero. Allora per uno dei giocatori (possiamo porre che sia Ali senza perdere il valore generale del ragionamento) deve esistere una deviazione vantaggiosa, ovvero una strategia MSF che produca un payoff più alto rispetto a INFLESSIBILE. Ora, questa strategia dovrebbe a un certo punto prevedere mosse diverse da quelle di

INFLESSIBILE, altrimenti realizzerebbe la medesima utilità. Quindi, a un certo punto dovrebbe giocare *testimoniare*. Ma a quel punto la strategia INFLESSIBILE di Bo reagirebbe passando alla modalità punitiva, ovvero scegliendo di testimoniare in modo permanente. A quel punto Ali sarebbe condannata a ricevere un payoff non maggiore di -5 , ovvero peggiore del -1 che avrebbe ottenuto scegliendo INFLESSIBILE. Quindi, la scelta di INFLESSIBILE da parte di entrambi costituisce un equilibrio di Nash per il dilemma del prigioniero ripetuto all'infinito, offrendo un esito razionalmente fondato impossibile da raggiungere nella versione del gioco a singolo turno.

Si tratta di un esempio di una classe generale di risultati chiamata **folk theorem di Nash** (o “teorema popolare di Nash”), che caratterizza i risultati che si fondano su equilibri di Nash nei giochi ripetuti all’infinito. Poniamo che il *valore di sicurezza* di un giocatore sia il migliore payoff che il giocatore può ottenere con certezza. Allora la forma generale del folk theorem di Nash è grosso modo la seguente: *ogni risultato che assegna a ciascun giocatore un payoff pari almeno al suo valore di sicurezza è sostenibile come equilibrio di Nash in un gioco ripetuto all’infinito*. Le strategia INFLESSIBILE è la chiave del folk theorem: la reciproca minaccia di rappresaglia contro chi mancasse di fare la propria parte per il risultato desiderato fa sì che i giocatori siano disciplinati. Ma funziona come deterrente solo se il giocatore crede che l’altro abbia adottato questa strategia, o che potrebbe averla adottata.

Possiamo ottenere soluzioni differenti anche cambiando gli agenti, invece di cambiare le regole di ingaggio. Supponiamo che gli agenti siano macchine a stati finiti con n stati e che giochino a un gioco con $m > n$ mosse in totale. Gli agenti non sono quindi in grado di rappresentare il numero delle mosse rimanenti e devono comportarsi come se fosse sconosciuto. Di conseguenza non possono applicare l’induzione a ritroso e nel dilemma del prigioniero ripetuto sono liberi di arrivare al più favorevole equilibrio (*rifiutare, rifiutare*). In questo caso, l’ignoranza è effettivamente un bene; o meglio, il fatto che l’avversario ci ritenga ignoranti è un bene. Il successo in questi giochi ripetuti dipende in misura significativa dall’essere *percepiti* dall’avversario come aggressivi o come sciocchi, non dalle proprie caratteristiche reali.

**folk theorem
di Nash**



18.2.4 Giochi sequenziali: la forma estesa

Nel caso generale, un gioco consiste in una sequenza di turni che non devono necessariamente essere tutti uguali. Il modo migliore per rappresentare questi giochi è un albero di gioco, che i teorici dei giochi chiamano **forma estesa**. L’albero contiene le medesime informazioni che abbiamo visto nel Paragrafo 5.1: uno stato iniziale S_0 , una funzione DEVE-MUOVERE(s) che indica a quale giocatore spetta la mossa, una funzione AZIONI(s) che enumera le azioni possibili, una funzione RISULTATO(s, a) che definisce la transizione a un nuovo stato e una funzione parziale UTILITÀ(s, p), definita solo per gli stati terminali, che assegna un payoff a ciascun giocatore. I giochi stocastici possono essere rappresentati introducendo un ulteriore giocatore, il *Caso*, che può eseguire azioni aleatorie. La “strategia” di *Caso* fa parte della definizione del gioco, ed è specificata come una distribuzione di probabilità sulle azioni (gli altri giocatori possono scegliere le proprie strategie). Per rappresentare i giochi con azioni non deterministiche, come il biliardo, scomponiamo l’azione in due parti: l’effettiva azione del giocatore ha un risultato deterministico, poi il turno passa a *Caso*, che reagisce all’azione nel proprio modo imprevedibile.

forma estesa

Per il momento, ricorreremo a un’ipotesi semplificatrice: che i giocatori dispongano di **informazione perfetta**. Informazione perfetta significa, approssimativamente, che quando il gioco richiede una decisione i giocatori sanno esattamente in quale punto dell’albero di gioco si trovano: non c’è incertezza riguardo a ciò che è accaduto in precedenza nel gioco. Questa è ovviamente la situazione in giochi come gli scacchi o il Go, ma non in giochi come il poker o il Kriegspiel. Nel prossimo paragrafo vedremo come utilizzare la forma estesa per rappre-

**informazione
perfetta**

sentare l'**informazione imperfetta** nei giochi; per il momento, però, assumeremo che vi sia informazione perfetta.

In un gioco in forma estesa con informazione perfetta, una strategia è una funzione che per ogni stato decisionale s indica al giocatore quale azione eseguire tra quelle di $AZIONI(s)$. Una volta che i giocatori hanno scelto le rispettive strategie, il profilo di strategie risultante tracerà un percorso attraverso l'albero di gioco, dallo stato iniziale S_0 a uno stato terminale, e la funzione UTILITÀ definisce l'utilità ottenuta da ciascun giocatore.

Data questa impostazione, possiamo applicare direttamente l'apparato degli equilibri di Nash che abbiamo introdotto in precedenza per analizzare i giochi in forma estesa. Per calcolare gli equilibri di Nash è possibile utilizzare una semplice generalizzazione della tecnica di ricerca minimax vista nel Capitolo 5. Nella letteratura riguardante i giochi in forma estesa, la tecnica è detta induzione a ritroso; l'abbiamo già utilizzata in modo informale per analizzare il dilemma del prigioniero con un numero finito di ripetizioni. L'induzione a ritroso utilizza la programmazione dinamica, procedendo all'indietro dagli stati terminali allo stato iniziale, associando mano a mano a ciascuno stato il profilo di payoff (l'assegnazione dei payoff ai giocatori) che si otterrebbe giocando in modo ottimo da quel punto in avanti.

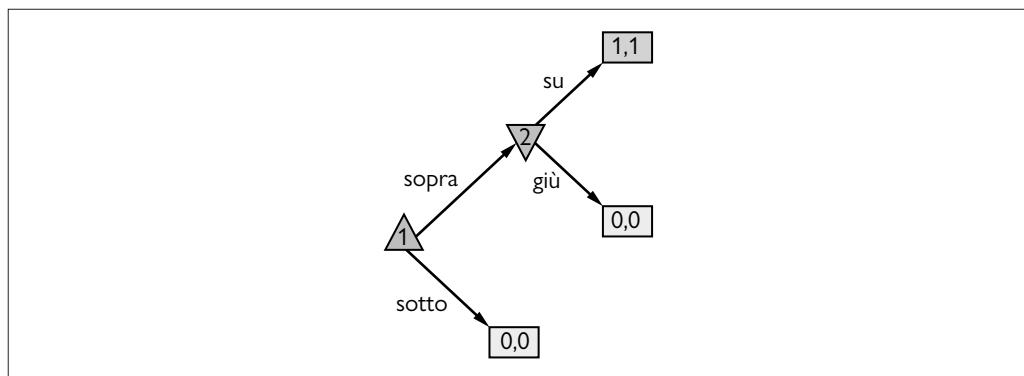
Nello specifico, per ciascuno stato non terminale s , se tutti i nodi discendenti da s sono associati a un profilo di payoff, allora si associa a s il profilo di payoff del nodo discendente che massimizza il payoff del giocatore che deve decidere nello stato s (se c'è equivalenza, si sceglie arbitrariamente; se si hanno nodi di casualità, si calcola l'utilità attesa). L'algoritmo di induzione a ritroso ha con certezza un termine; inoltre, viene eseguito in tempo polinomiale nella dimensione dell'albero di gioco.

A mano a mano che l'algoritmo compie il proprio lavoro, delinea le strategie per ciascun giocatore. Risulta che tali strategie sono strategie in equilibrio di Nash e che il profilo di payoff associato allo stato iniziale è un profilo che verrebbe ottenuto giocando strategie in equilibrio di Nash. Perciò, nei giochi in forma estesa le strategie in equilibrio di Nash possono essere calcolate in tempo polinomiale ricorrendo all'induzione a ritroso; e poiché è certo che l'algoritmo assegna un profilo di payoff allo stato iniziale, allora ogni gioco in forma estesa ha almeno un equilibrio di Nash con strategie pure.

Si tratta di conclusioni interessanti, ma vi sono diverse puntualizzazioni da fare. Gli alberi di gioco assumono molto rapidamente dimensioni molto grandi, quindi in questo contesto è necessario comprendere cosa significa tempo polinomiale di esecuzione. Ancor più problematicamente, l'equilibrio di Nash stesso ha alcune limitazioni quando è applicato a giochi in forma estesa. Consideriamo il gioco della Figura 18.4. Il giocatore 1 ha a disposizione due mosse: *sopra* o *sotto*. Se sceglie *sotto*, al giocatore 2 si presenta una scelta tra *su* e *giù*: se sceglie *giù*, entrambi i giocatori ricevono il payoff 0, mentre se sceglie *su* entrambi ricevono 1.

L'induzione a ritroso ci dice immediatamente che (*sopra*, *su*) è un equilibrio di Nash, nel quale entrambi i giocatori ricevono un payoff pari a 1. Tuttavia, anche (*sotto*, *giù*) è un equi-

Figura 18.4
Un gioco in forma estesa con un equilibrio di Nash controintuitivo.



librio di Nash, nel quale entrambi ricevono un payoff pari a 0. Il giocatore 2 minaccia il giocatore 1 segnalando che, se fosse chiamato a decidere, sceglierrebbe *giù*, determinando per il giocatore 1 il payoff 0; in questo caso, il giocatore 1 non ha migliore alternativa che scegliere *sotto*. Il problema è che la minaccia del giocatore 2 (di giocare *giù*) non è una **minaccia credibile**, perché se il giocatore 2 dovesse effettivamente compiere una scelta, sceglierrebbe *su*.

Una variante dell'equilibrio di Nash chiamata **equilibrio perfetto di Nash nei sottogiochi** si occupa di questo problema. Per definirlo, occorre il concetto di **sottogioco**. Ogni stato decisionale di un albero di gioco (compreso lo stato iniziale) definisce un sottogioco. Il gioco della Figura 18.4 contiene quindi due sottogiochi: uno con radice nello stato in cui la decisione spetta al giocatore 1, l'altro con radice in quello dove decide il giocatore 2. *Un profilo di strategie costituisce un equilibrio perfetto di Nash in un gioco G se è un equilibrio di Nash in ogni sottogioco di G.* Applicando questa definizione al gioco della Figura 18.4, troviamo che (*sopra, su*) è perfetto nei sottogiochi, mentre (*sotto, giù*) non lo è, perché la scelta *giù* non è un equilibrio di Nash per il sottogioco che ha radice nello stato in cui deve decidere il giocatore 2.

Per definire l'equilibrio perfetto di Nash nei sottogiochi occorre una nuova terminologia ma non occorre alcun nuovo algoritmo. Le strategie calcolate mediante l'induzione a ritroso sono equilibri perfetti di Nash nei sottogiochi; ne consegue che ogni gioco in forma estesa con informazione perfetta ha un equilibrio perfetto di Nash nei sottogiochi, che può essere calcolato in tempo polinomiale nella dimensione dell'albero di gioco.

minaccia credibile

equilibrio perfetto di Nash nei sottogiochi



Caso e mosse simultanee

Per rappresentare in forma estesa i giochi stocastici, come il backgammon, si aggiunge un giocatore chiamato *Caso*, le cui scelte sono determinate da una distribuzione di probabilità.

Per rappresentare le mosse simultanee, come quelle del dilemma del prigioniero o della morra a due dita, stabiliamo arbitrariamente un ordine di gioco per i giocatori, aggiungendo però che le azioni di chi gioca prima non sono osservabili dai giocatori successivi: per esempio, Ali deve scegliere *rifiutare* o *testimoniare* prima di Bo, ma Bo non conosce la scelta di Ali nel momento in cui tocca a lui scegliere (possiamo anche rappresentare il fatto che la mossa viene rivelata in seguito). Supponiamo invece che i giocatori ricordino sempre tutte le *loro* azioni precedenti; questa ipotesi è detta di **memoria perfetta**.

Modellare l'informazione imperfetta

Una caratteristica fondamentale della forma estesa, che la distingue dagli alberi di gioco che abbiamo visto nel Capitolo 5, è che può considerare l'osservabilità parziale. I teorici dei giochi utilizzano il termine **informazione imperfetta** per descrivere situazioni in cui i giocatori non hanno informazioni certe sullo stato del gioco.

informazione imperfetta

Purtroppo, l'induzione a ritroso non funziona con i giochi a informazione imperfetta, i quali, in generale, sono anche notevolmente più complessi da risolvere dei giochi a informazione perfetta.

Nel Paragrafo 5.6 abbiamo visto che un giocatore in un gioco parzialmente osservabile come Kriegspiel può creare un albero di gioco sullo spazio degli **stati-credenza**. Con tale albero, come abbiamo visto, in alcuni casi un giocatore può trovare una sequenza di mosse (una strategia) che conduce a uno scacco matto indipendentemente dallo stato da cui si è partiti e dalla strategia adottata dall'avversario. Le tecniche del Capitolo 5, invece, non sono in grado di indicare a un giocatore che cosa fare quando non c'è uno scacco matto garantito. Se la migliore strategia dipende dalla strategia dell'avversario e viceversa, il metodo minimax (o alfa-beta) di per sé non è in grado di trovare una soluzione. La forma estesa *consente* di trovare soluzioni perché rappresenta anche gli stati-credenza (i teorici li chiamano **insiemi informativi**) di tutti i giocatori contemporaneamente. Da tale rappresentazione possiamo ricavare soluzioni di equilibrio, come abbiamo fatto con i giochi in forma normale.

insieme informativo

Come semplice esempio di gioco sequenziale, collichiamo due agenti nel mondo 4×3 della Figura 17.1 (Capitolo 17) e facciamoli muovere simultaneamente fino a quando uno degli agenti non raggiunge una casella di uscita ottenendo il payoff corrispondente. Se specifichiamo che non avviene alcun movimento se i due agenti tentano di entrare contemporaneamente nella stessa casella (un problema comune a molti incroci stradali), allora alcune strategie pure possono condurre a uno stallo. In questo gioco, quindi, per ottenere buoni risultati gli agenti necessitano di una strategia mista, ovvero di scegliere casualmente tra andare avanti e rimanere fermi. È esattamente ciò che si fa per risolvere il problema della collisione tra pacchetti nelle reti Ethernet.

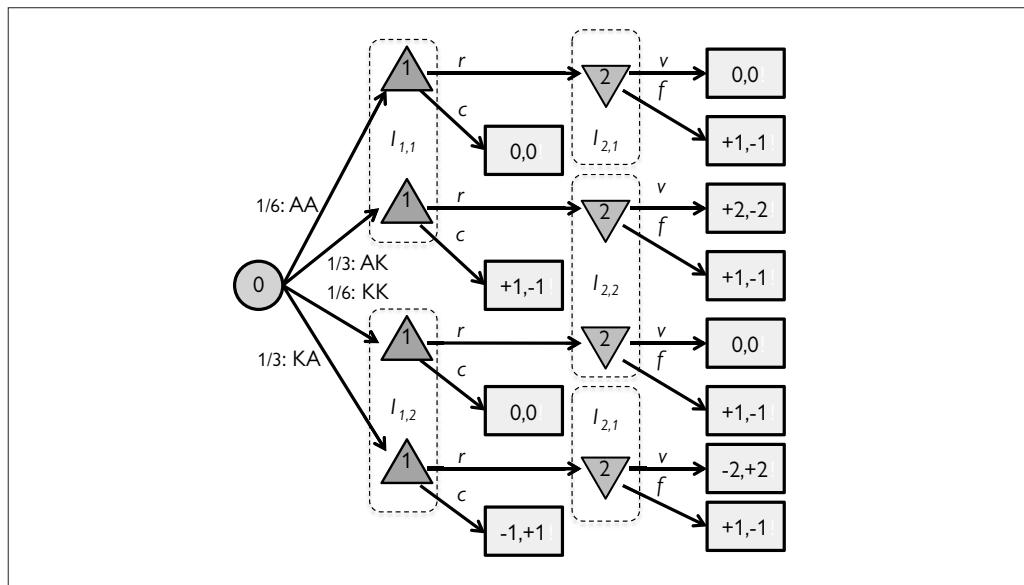
Consideriamo ora una semplicissima variante del poker. Il mazzo è composto solo da quattro carte, due assi e due re. A ciascun giocatore viene consegnata una carta. Il primo giocatore ha l'opzione di *rilanciare*, alzando la posta da 1 a 2 punti, oppure di passare senza puntare (*cip*). Se il giocatore 1 passa, il gioco termina. Se il giocatore 1 rilancia, allora il giocatore 2 ha la possibilità di *vedere*, accettando che il gioco valga due punti, o di abbandonare (*fold*), concedendo 1 punto. Se il gioco non termina con un abbandono, allora il payoff dipende dalle carte: è zero per entrambi se hanno la stessa carta; altrimenti il giocatore con il re paga la posta al giocatore che ha l'asso.

L'albero in forma estesa di questo gioco è mostrato nella Figura 18.5. Il giocatore 0 è il *Caso*; i giocatori 1 e 2 sono indicati da triangoli. Ogni azione è rappresentata da una freccia con una lettera, corrispondente a *rilancio*, *cip*, *vedo* o *fold*; oppure, per il giocatore *Caso*, alle quattro possibili distribuzioni di carte (“AK” significa che il giocatore 1 riceve un asso e il giocatore 2 un re). Gli stati terminali sono rettangoli contrassegnati dai payoff dei due giocatori. Gli insiemi informativi sono rappresentati come riquadri tratteggiati; per esempio, $I_{1,1}$ è l'insieme informativo in cui il giocatore 1 è di turno e sa di avere un asso (ma non conosce la carta del giocatore 2). Nell'insieme informativo $I_{2,1}$, è il turno del giocatore 2, che sa di avere un asso e sa che il giocatore 1 ha rilanciato, ma non conosce la carta di quest'ultimo (poiché la pagina ha solamente due dimensioni, questo insieme è rappresentato con due riquadri invece di uno).

Un modo per risolvere un gioco in forma estesa consiste nel convertirlo alla forma normale. Ricordiamo che la forma normale è una matrice, nella quale ogni riga è associata a una strategia pura per il giocatore 1 e ogni colonna a una strategia pura per il giocatore 2. In un

Figura 18.5

Forma estesa di una versione semplificata del poker, con due giocatori e soltanto quattro carte. Le mosse sono *r* (rilancio), *f* (fold), *v* (vedo), e *c* (cip).



gioco in forma estesa una strategia pura per il giocatore i corrisponde a un'azione per ogni insieme informativo che coinvolge quel giocatore. Nella Figura 18.5, quindi, una strategia pura per il giocatore 1 è “rilancio nella situazione $I_{1,1}$ (ovvero, quando ho un asso) e *cip* nella situazione $I_{1,2}$ (quando ho un re)”. Nella seguente matrice dei payoff, questa strategia è chiamata *rc*. Analogamente, la strategia *vf* per il giocatore 2 significa “vedo quando ho un asso e abbandono (*fold*) quando ho un re”. Dato che questo è un gioco a somma zero, la matrice indica solamente il payoff del giocatore 1; il giocatore 2 ha sempre il payoff opposto:

	2: vv	2: vf	2: ff	2: fv
1: rr	0	-1/6	1	7/6
1: cr	-1/3	-1/6	5/6	2/3
1: rc	1/3	0	1/6	1/2
1: cc	0	0	0	0

Questo gioco è tanto semplice da avere due equilibri con strategie pure, indicati in grassetto: *vf* per il giocatore 2 e *rc* oppure *cc* per il giocatore 1. In generale è possibile risolvere i giochi estesi portandoli alla forma normale e poi cercando una soluzione (solitamente una strategia mista) con i metodi standard di programmazione lineare. In teoria funziona. Se però un giocatore ha I insiemi informativi e a azioni per ogni insieme, allora avrà a^I strategie pure. In altre parole, la dimensione della matrice della forma normale è esponenziale nel numero di insiemi informativi, quindi in pratica l'approccio funziona solo per alberi di gioco di piccole dimensioni, con circa una dozzina di stati. Un gioco come il poker Texas hold ‘em ha circa 10^{18} stati, il che rende questo approccio del tutto impraticabile.

Quali sono le alternative? Nel Capitolo 5 abbiamo visto che la ricerca alfa-beta è in grado di gestire giochi con informazione perfetta e alberi di gioco molto grandi, generando l'albero progressivamente, eliminando alcuni rami e valutando euristicamente i nodi non terminali. Tale approccio però non funziona bene con i giochi con informazione imperfetta, per due ragioni: primo, eliminare rami è più difficile, perché si ha a che fare con strategie miste che combinano più rami, non con strategie pure che scelgono sempre il ramo migliore. Secondo, è più difficile valutare euristicamente un nodo non terminale, perché si ha a che fare con un insieme informativo, non con singoli stati.

Koller *et al.* (1996) vengono in aiuto con una rappresentazione alternativa dei giochi in forma estesa detta **forma sequenziale**, che è lineare e non esponenziale nella dimensione dell'albero. Invece di rappresentare strategie, essa rappresenta dei percorsi attraverso l'albero; il numero di percorsi è uguale al numero dei nodi terminali. Anche a questa rappresentazione si possono applicare metodi standard di programmazione lineare. Il sistema risultante è in grado di risolvere varianti del poker con 25.000 stati in uno o due minuti. È un'accelerazione esponenziale rispetto all'approccio della forma normale ma siamo ancora lontani dal poter gestire, per esempio, il Texas hold ‘em a due giocatori, con i suoi 10^{18} stati.

forma sequenziale

Se non possiamo gestire 10^{18} stati, forse possiamo semplificare il problema modificando il gioco per ottenere una forma più semplice. Per esempio, se ho un asso e sto considerando la possibilità che la prossima carta mi dia una coppia di assi, allora non mi interessa il seme della prossima carta; secondo le regole del poker ogni seme è altrettanto valido. Ciò suggerisce un'**astrazione** del gioco, nella quale i semi vengano ignorati. L'albero di gioco risultante sarà più piccolo di un fattore $4!=24$. Supponiamo di poter risolvere questo gioco ridotto; che relazione c'è tra la soluzione di questo gioco e il gioco originale? Se nessun giocatore intende realizzare un colore (l'unica combinazione in cui il seme è rilevante), allora la soluzione dell'astrazione sarà una soluzione anche per il gioco originale. Se invece uno dei giocatori vuole tentare il colore, allora l'astrazione sarà una soluzione approssimativa (ma è possibile calcolare dei limiti per l'errore).

Esistono molte possibilità di astrazione. Per esempio, al punto del gioco in cui ogni giocatore ha due carte, se io ho una coppia di regine allora le combinazioni di carte degli altri giocatori si possono astrarre in tre classi: *migliori* (una coppia di re o una coppia di assi), *uguali* (coppia di regine) o *peggiori* (qualsiasi altra cosa). Questa astrazione potrebbe però essere troppo grossolana. Un’astrazione migliore potrebbe suddividere ulteriormente la classe *peggiori* nelle classi *coppia media* (dai nove fino ai jack), *coppia bassa*, e *nessuna coppia*. Questi esempi sono astrazioni di stati; è possibile astrarre anche le azioni. Per esempio, invece di avere un’azione di puntata per ogni intero da 1 a 1000, potremmo ammettere solamente le puntate $10^0, 10^1, 10^2$ e 10^3 . Oppure potremmo eliminare completamente uno dei giri di puntate. Possiamo definire astrazioni anche sui nodi di casualità, considerando solamente un sottoinsieme delle possibili distribuzioni di carte. Ciò equivale alla tecnica del rollout utilizzata nei programmi di Go. Combinando tra loro tutte queste astrazioni, possiamo ridurre i 10^{18} stati del poker a 10^7 , una dimensione che consente di trovare una soluzione con le tecniche attuali.

Nel Capitolo 5 abbiamo visto che programmi di poker come Libratus e DeepStack sono stati in grado di sconfiggere campioni umani nel Texas hold ‘em a due giocatori. Più recentemente, il programma Pluribus è riuscito a sconfiggere campioni umani nel poker a sei giocatori, in due varianti: cinque copie del programma al tavolo con un umano, e una copia del programma con cinque umani. Ciò significa un enorme incremento della complessità. Con un solo avversario, esistono $\binom{50}{2} = 1225$ possibilità per le carte nascoste dell’avversario. Con cinque avversari esistono invece $\binom{50}{10} \approx 10$ miliardi di possibilità. Pluribus sviluppa una strategia di base giocando da solo, poi la modifica durante le partite vere e proprie reagendo alle specifiche situazioni. Pluribus utilizza una combinazione di tecniche, tra cui la ricerca ad albero Monte Carlo, la ricerca a profondità limitata e l’astrazione.

La forma estesa è una rappresentazione versatile: può gestire ambienti parzialmente osservabili, multiagente, stocastici, sequenziali e in tempo reale: la maggior parte dei casi difficili dell’elenco di proprietà degli ambienti del Paragrafo 2.3.2. Esistono però due limiti, per la forma estesa in particolare e per la teoria dei giochi in generale. In primo luogo, essa non funziona bene con gli stati e le azioni continui (sebbene siano state proposte alcune estensioni per il caso continuo; per esempio, la teoria dell’**oligopolio di Cournot** ricorre alla teoria dei giochi per risolvere problemi in cui due aziende scelgono i prezzi dei loro prodotti in uno spazio continuo). Secondo, nella teoria dei giochi si assume che il gioco sia *noto*. Parti del gioco possono essere definite come non osservabili per alcuni dei giocatori, ma si deve sapere quali parti non sono osservabili. Nei casi in cui i giocatori apprendono la struttura ignota del gioco gradualmente, il modello inizia a non reggere più. Esaminiamo allora le fonti di incertezza e la possibilità di rappresentarle nella teoria dei giochi.

Azioni: non esiste un modo semplice per rappresentare un gioco in cui i giocatori devono scoprire le azioni disponibili. Consideriamo il gioco tra gli autori di virus informatici e gli esperti di sicurezza. Parte del problema consiste nel prevedere quali saranno le prossime mosse degli autori di virus.

Strategie: la teoria dei giochi è molto valida per rappresentare l’idea che le strategie degli altri giocatori siano inizialmente sconosciute, a patto di assumere che tutti gli agenti siano razionali. La teoria non dà indicazioni per il caso in cui gli altri giocatori non sono pienamente razionali. La nozione di **equilibrio di Bayes–Nash** affronta parzialmente questo problema: è un equilibrio rispetto alla distribuzione di probabilità a priori di un giocatore sulle strategie degli altri giocatori; in altre parole, esprime le credenze di un giocatore circa le probabili strategie degli altri.

Casualità: se un gioco dipende dal lancio di un dado, è abbastanza semplice introdurre nel modello un nodo di casualità con una distribuzione uniforme dei risultati. Ma cosa accade se si ammette l’eventualità che il dado sia truccato? Possiamo rappresentare questa situazione con un altro nodo di casualità, più in alto nell’albero, con due rami corrispondenti

a “dado non truccato” e a “dado truccato”, in modo tale che i nodi corrispondenti nei due rami facciano parte dello stesso insieme informativo (ovvero, i giocatori non sanno se il dado sia truccato o meno). E se sospettassimo che l'avversario invece lo sappia? Allora potremmo aggiungere *un altro* nodo di casualità, con un ramo che rappresenta il caso in cui l'avversario sa e l'altro ramo per il caso in cui non sa.

Utilità: che cosa dobbiamo fare se non conosciamo le utilità dell'avversario? Anche questo caso può essere modellato con un nodo di casualità, tale che in ogni ramo l'altro agente conosca le proprie utilità, che noi invece non conosciamo. Ma cosa accade se non conosciamo le *nostre* utilità? Per esempio, come posso sapere se è razionale ordinare l'insalata dello chef, se non so quanto mi piacerà? Possiamo inserire questo caso nel modello per mezzo di un ulteriore nodo di casualità, che specifichi una inosservabile “qualità intrinseca” dell'insalata.

Vediamo quindi che la teoria dei giochi è valida per rappresentare la maggior parte delle fonti di incertezza, al prezzo però di raddoppiare le dimensioni dell'albero ogni volta che si aggiunge un nodo; ciò che conduce rapidamente ad alberi di dimensioni intrattabili. Per questo e per altri problemi, la teoria dei giochi è stata utilizzata principalmente per *analizzare* gli ambienti che si trovano in equilibrio, piuttosto che per *controllare* gli agenti all'interno di un ambiente.

18.2.5 Payoff incerti e giochi di assistenza

Nel Capitolo 1 abbiamo parlato dell'importanza di progettare sistemi di IA che riescano a operare quando vi è incertezza riguardo al reale obiettivo umano. Nel Capitolo 16 abbiamo introdotto un semplice modello per l'incertezza riguardo alle *proprie* preferenze, mediante l'esempio del gelato al gusto durian. Con il semplice accorgimento di aggiungere al modello una nuova variabile latente per rappresentare le preferenze incerte, oltre a un adeguato modello sensoriale (per esempio percepire il gusto di un piccolo campione di gelato), le preferenze incerte possono essere gestite in modo naturale.

Nel Capitolo 16 è stato esaminato anche il **problema del pulsante di spegnimento**: abbiamo mostrato che un robot con conoscenza incerta riguardo alle preferenze umane si rimette alla persona e accetta di spegnersi. In questo problema, il robot Robbie non ha conoscenza certa delle preferenze della persona Harriet, ma la decisione di Harriet (spegnere o non spegnere Robbie) è modellata come una semplice conseguenza deterministica delle sue preferenze rispetto all'azione proposta dal robot. Generalizziamo ora questa idea per un gioco a due persone, detto **gioco di assistenza**, nel quale Harriet e Robbie sono entrambi giocatori. Assumiamo che Harriet osservi le proprie preferenze θ e che agisca conformemente a esse, mentre Robbie ha una probabilità a priori $P(\theta)$ sulle preferenze di Harriet. Il payoff è definito da θ ed è identico per i due giocatori: Harriet e Robbie massimizzano entrambi il payoff di Harriet. In questo modo, il gioco di assistenza fornisce un modello formale del concetto di IA con benefici dimostrabili, presentato nel Capitolo 1.

Oltre al comportamento deferente mostrato da Robbie nel problema dello spegnimento (una versione ridotta dei giochi di assistenza), nei giochi di assistenza emergono come strategie di equilibrio anche altri comportamenti: le azioni di Harriet che potremmo descrivere come insegnare, premiare, comandare, correggere, mostrare o spiegare, e le azioni di Robbie che descriveremmo come chiedere il permesso, imparare dall'esempio, dedurre le preferenze e così via. Il punto chiave è che questi comportamenti non devono essere pianificati: risolvendo il gioco, Harriet e Robbie determinano da soli come far fluire le informazioni sulle preferenze da Harriet a Robbie, di modo che Robbie possa essere più utile a Harriet. Non occorre stabilire a priori che Harriet debba “dare ricompense” o che Robbie debba “seguire le istruzioni”, sebbene queste siano interpretazioni ragionevoli di come finiranno per comportarsi.

Per illustrare i giochi di assistenza utilizzeremo il **gioco delle graffette**. È un gioco molto semplice in cui la persona Harriet è incentivata a “segnalare” al robot Robbie alcune infor-

gioco delle graffette

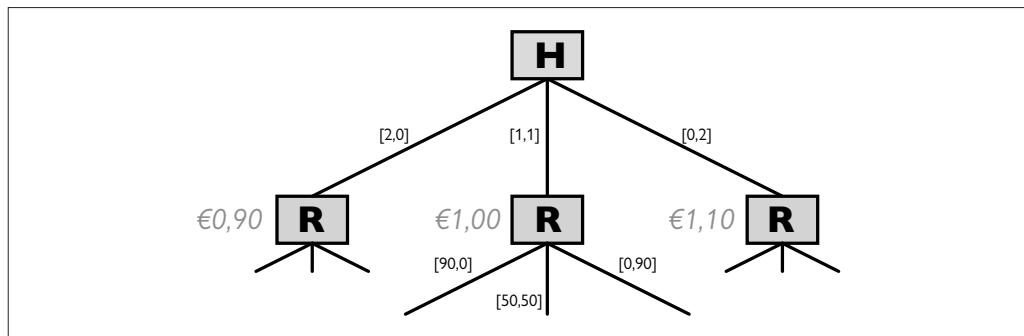


Figura 18.6 Il gioco delle graffette. Ogni ramo è marcato $[f, g]$ a indicare il numero di fermagli e di graffette fabbricati in quel caso. La persona Harriet può scegliere di fabbricare due fermagli, due graffette oppure uno di ciascuno (i valori in corsivo verde sono i valori per Harriet se il gioco finisce in quel punto, assumendo $\theta = 0,45$). Il robot Robbie può poi scegliere di fabbricare 90 fermagli, 90 graffette o 50 di ciascuno.

mazioni sulle proprie preferenze. Robbie è capace di interpretare quel segnale perché sa risolvere il gioco e quindi è in grado di capire come dovrebbero essere le vere preferenze di Harriet per far sì che emetta un segnale come quello.

I passaggi del gioco sono illustrati nella Figura 18.6. Si tratta di fabbricare fermagli e graffette. Le preferenze di Harriet sono espresse da una funzione di payoff che dipende dal numero di fermagli e dal numero di graffette fabbricati, con un certo “tasso di cambio” tra i due prodotti. Il parametro di preferenza di Harriet, θ , indica il valore relativo (in euro) di un fermaglio; per esempio, Harriet potrebbe valutare i fermagli $\theta = 0,45$ euro, che significa che le graffette valgono $1 - \theta = 0,55$ euro. Quindi, se vengono prodotti f fermagli e g graffette, il payoff di Harriet sarà $p\theta + g(1 - \theta)$ euro in totale. La probabilità a priori di Robbie è $P(\theta) = \text{Uniforme}(\theta; 0, 1)$. Nel gioco, Harriet muove per prima e può scegliere di fabbricare due fermagli, due graffette o uno di ognuno. Successivamente Robbie può scegliere di fabbricare 90 fermagli, 90 graffette o 50 di ognuno.

Notate che, se giocasse da sola, Harriet fabbricherebbe solamente due graffette, per un valore di €1,10 (cfr. le indicazioni al primo livello dell’albero nella Figura 18.6). Ma Robbie osserva e impara dalla sua scelta. Che cosa impara, esattamente? Dipende da come Harriet effettua la propria scelta. In che modo Harriet effettua la scelta? Dipende da come Robbie la interpreterà. Possiamo risolvere questa circolarità trovando un equilibrio di Nash. In questo caso, l’equilibrio è unico e si può trovare applicando il metodo della miglior risposta miope: scegliere una strategia qualsiasi per Harriet; scegliere la strategia migliore per Robbie data quella di Harriet; scegliere la migliore strategia per Harriet data quella di Robbie, e così via. Il processo si sviluppa nel modo seguente.

1. Iniziamo con la strategia più soddisfacente per Harriet: fabbricare due fermagli se preferisce i fermagli; fabbricare un oggetto per tipo se è indifferente; fabbricare due graffette se preferisce le graffette.
2. Data questa strategia di Harriet, Robbie deve considerare tre possibilità:
 - (a) Se Robbie vede Harriet fabbricare due fermagli, inferisce che Harriet preferisca i fermagli, e quindi che il valore di un fermaglio sia uniformemente distribuito tra 0,5 e 1, con una media di 0,75. In questo caso, il piano migliore per lui consiste nel fabbricare 90 fermagli, per un valore atteso per Harriet pari a €67,50.
 - (b) Se Robbie vede Harriet fabbricare un oggetto per tipo, inferisce che Harriet valuti 0,50 sia i fermagli che le graffette, quindi la scelta migliore è fabbricarne 50 di ognuno.
 - (c) Se Robbie vede Harriet fabbricare due graffette, per lo stesso ragionamento del punto (a) dovrebbe fabbricare 90 graffette.

3. Data questa strategia di Robbie, la migliore strategia per Harriet ora è un po' differente rispetto a quella del passaggio 1. Se Robbie risponderà alla sua scelta di fabbricare un oggetto per tipo fabbricandone a propria volta 50 per tipo, allora per Harriet è più vantaggioso fabbricare un oggetto per tipo non solamente se è esattamente indifferente tra i due, ma anche se la sua preferenza si trova in un punto vicino all'indifferenza. Nello specifico, ora la politica ottima per Harriet è fabbricare un oggetto per tipo se valuta che i fermagli abbiano un valore compreso tra circa 0,446 e 0,554.
4. Data questa nuova strategia per Harriet, la strategia di Robbie rimane invariata. Per esempio, se Harriet sceglie uno di ognuno, Robbie inferisce che il valore di un fermaglio è uniformemente distribuito tra 0,446 e 0,554, con una media di 0,50, quindi la scelta migliore è 50 di ognuno. Poiché la strategia di Robbie è la stessa del passaggio 2, la risposta migliore di Harriet sarà la stessa del passaggio 3, quindi abbiamo trovato un equilibrio.

Con la propria strategia Harriet sta in effetti insegnando a Robbie le proprie preferenze utilizzando un semplice codice, o se si preferisce un linguaggio, che emerge dall'analisi dell'equilibrio. Notate inoltre che Robbie non impara mai le preferenze di Harriet in modo esatto ma ne impara quanto basta per agire in modo ottimo per suo conto. Agisce cioè nello stesso modo in cui agirebbe se conoscesse le preferenze di Harriet in modo esatto. Robbie ha benefici dimostrabili per Harriet dati i presupposti enunciati e assumendo che Harriet giochi correttamente.

La miglior risposta miope funziona per questo esempio e per altri analoghi, ma non in casi più complessi. È possibile provare che, quando non vi siano vincoli che provocano problemi di coordinamento, la ricerca di un profilo di strategie ottime per un gioco di assistenza è riducibile alla soluzione di un POMDP il cui spazio degli stati è lo spazio degli stati sottostante il gioco, più i parametri di preferenza θ delle persone. In generale i POMDP sono molto difficili da risolvere (cfr. Paragrafo 17.5) ma quelli che rappresentano giochi di assistenza hanno una struttura aggiuntiva che consente algoritmi più efficienti.

I giochi di assistenza possono essere generalizzati per consentire la partecipazione di più umani, più robot, umani con razionalità imperfetta, umani che non conoscono le proprie preferenze e così via. Se si fornisce uno spazio di azione fattorizzato o strutturato, invece delle semplici azioni atomiche del gioco dei fermagli, le opportunità di comunicazione possono essere notevolmente migliorate. Finora sono state esplorate poche di queste varianti, ma ci aspettiamo che la proprietà fondamentale dei giochi di assistenza rimanga valida: più intelligente è il robot, migliore sarà il risultato per l'essere umano.

18.3 Teoria dei giochi cooperativi

Ricordiamo che i giochi cooperativi riproducono scenari decisionali in cui gli agenti possono stipulare tra loro accordi vincolanti per cooperare. Ciò consente loro di ottenere benefici maggiori rispetto a quelli che otterrebbero agendo separatamente.

Iniziamo introducendo un modello per una classe di **giochi cooperativi**. Formalmente, questi giochi sono chiamati “giochi cooperativi con utilità trasferibile in forma di funzione caratteristica”. L’idea del modello è che, quando un gruppo di agenti coopera, il gruppo nel suo complesso ottiene un certo valore di utilità, che può essere poi ripartito tra i suoi membri. Il modello non dice quali saranno le azioni compiute dagli agenti, né la struttura del gioco specifica come il valore ottenuto verrà suddiviso (a questo si arriverà in seguito).

Formalmente, utilizziamo la formula $G = (N, \nu)$ per indicare che un gioco cooperativo G è definito da un insieme di giocatori $N = \{1, \dots, n\}$ e da una **funzione caratteristica** ν che, per ogni sottoinsieme di giocatori $C \subseteq N$, indica il valore che quel gruppo di giocatori potrebbe ottenere se i suoi membri scegliessero di cooperare.

**funzione
caratteristica**

Generalmente si assume che l’insieme di giocatori vuoto ottenga un valore nullo ($\nu(\{\}) = 0$) e che la funzione sia non negativa ($\nu(C) \geq 0$ per ogni C). In alcuni giochi si assume inoltre che agendo da soli i giocatori non ottengano nulla: $\nu(\{i\}) = 0$ per ogni $i \in N$.

18.3.1 Strutture di coalizioni e risultati

coalizione

È uso comune riferirsi a un sottoinsieme di giocatori C come a una **coalizione**. Nel linguaggio comune, “coalizione” indica un insieme di soggetti che condividono una causa; noi invece chiameremo coalizione un sottoinsieme di giocatori *qualsiasi*. L’insieme N di tutti i giocatori è detto **grande coalizione**.

Nel nostro modello, ogni giocatore deve scegliere di aderire a una e una sola coalizione (che può anche essere costituita unicamente dal giocatore stesso). Le coalizioni costituiscono quindi una **partizione** dell’insieme dei giocatori. Chiamiamo tale partizione **struttura di coalizioni**. Formalmente, una struttura di coalizioni per un insieme di giocatori N è un insieme di coalizioni $\{C_1, \dots, C_k\}$ tale che:

$$\begin{aligned} C_i &\neq \emptyset \\ C_i &\subseteq N \\ C_i \cap C_j &= \emptyset \text{ per ogni } i \neq j \in N \\ C_1 \cup \dots \cup C_k &= N. \end{aligned}$$

Per esempio, se abbiamo $N = \{1, 2, 3\}$, le coalizioni possibili sono sette:

$$\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{3, 1\} \text{ e } \{1, 2, 3\}$$

e le strutture di coalizioni possibili sono cinque:

$$\{\{1\}, \{2\}, \{3\}\}, \{\{1\}, \{2, 3\}\}, \{\{2\}, \{1, 3\}\}, \{\{3\}, \{1, 2\}\}, \text{ e } \{\{1, 2, 3\}\}.$$

Utilizziamo la notazione $CS(N)$ per indicare l’insieme di tutte le strutture di coalizioni sull’insieme di giocatori N , e $CS(i)$ per indicare la coalizione a cui appartiene il giocatore i .

Il **risultato** (*outcome*) di un gioco è definito dalle scelte che i giocatori fanno, riguardo a quali coalizioni formare e a come suddividere il valore $\nu(C)$ che ciascuna coalizione riceve. Formalmente, dato un gioco cooperativo definito da (N, ν) , il risultato è una coppia (CS, \mathbf{x}) costituita da una struttura di coalizioni e da un **vettore dei payoff** $\mathbf{x} = (x_1, \dots, x_n)$ dove x_i è il valore che va al giocatore i . Il payoff deve soddisfare il vincolo che ogni coalizione C ripartisca interamente il valore $\nu(C)$ tra i propri membri:

$$\sum_{i \in C} x_i = \nu(C) \quad \text{per ogni } C \in CS$$

Per esempio, dato il gioco $(\{1, 2, 3\}, \nu)$ dove $\nu(\{1\}) = 4$ e $\nu(\{2, 3\}) = 10$, un possibile risultato è: $(\{\{1\}, \{2, 3\}\}, (4, 5, 5))$.

Ovvero, il giocatore 1 rimane da solo e accetta il valore 4, mentre i giocatori 2 e 3 si uniscono per ricevere il valore 10, che poi scelgono di dividere in parti uguali.

Alcuni giochi cooperativi hanno la caratteristica che, quando due coalizioni si fondono, ottengono un risultato non peggiore di quello che avrebbero ottenuto rimanendo separate. Questa proprietà è detta **superadditività**. Formalmente, un gioco è superadditivo se la sua funzione caratteristica soddisfa la condizione seguente:

$$\nu(C \cup D) \geq \nu(C) + \nu(D) \quad \text{per ogni } C, D \subseteq N$$

Se un gioco è superadditivo, allora la grande coalizione riceve un valore che è pari o superiore al totale ricevuto da ogni altra struttura di coalizioni. Tuttavia, come vedremo tra poco, i giochi superadditivi non si concludono sempre con una grande coalizione, per lo stesso mo-

vettore dei payoff

superadditività

tivo per cui nel dilemma del prigioniero i giocatori non arrivano sempre a un esito Pareto-ottimo collettivamente desiderabile.

18.3.2 Strategia nei giochi cooperativi

Nella teoria dei giochi cooperativi, l'assunto base è che i giocatori scelgano strategicamente gli attori con cui cooperare. Come è intuitibile, non vorranno lavorare con giocatori improduttivi e cercheranno naturalmente dei giocatori che producano collettivamente un valore elevato per la coalizione. Ma i giocatori desiderati faranno a loro volta dei ragionamenti strategici. Prima di descrivere questi ragionamenti, occorrono altre definizioni.

Un'**imputazione** per un gioco cooperativo (N, v) è un vettore di payoff che soddisfa le due condizioni seguenti:

imputazione

$$\begin{aligned}\sum_{i=1}^n x_i &= v(N) \\ x_i &\geq v(\{i\}) \text{ per ogni } i \in N.\end{aligned}$$

La prima condizione stabilisce che un'imputazione deve distribuire il valore totale della grande coalizione; la seconda, nota come **razionalità individuale**, che ogni giocatore riceve almeno quanto avrebbe ricevuto lavorando da solo.

razionalità individuale

Data un'imputazione $\mathbf{x} = (x_1, \dots, x_n)$ e una coalizione $C \subseteq N$, definiamo $x(C)$ come la somma $\sum_{i \in C} x_i$: l'importo totale attribuito a C dall'imputazione \mathbf{x} .

nucleo

Definiamo poi **nucleo** di un gioco (N, v) l'insieme di tutte le imputazioni \mathbf{x} che soddisfano la condizione $x(C) \geq v(C)$ per ogni possibile coalizione $C \subseteq N$. Quindi, se un'imputazione \mathbf{x} non fa parte del nucleo, allora esiste una coalizione $C \subseteq N$ tale che $v(C) > x(C)$. I giocatori di C rifiuterebbero di partecipare alla grande coalizione perché starebbero meglio con C .

Il nucleo di un gioco è costituito quindi da tutti i possibili vettori di payoff che nessuna coalizione potrebbe rifiutare con la motivazione di poter ottenere di più rimanendo fuori dalla grande coalizione. Perciò, se il nucleo è vuoto, la grande coalizione non può formarsi, perché indipendentemente da come il payoff viene distribuito, qualche coalizione preferirà non partecipare. Le principali domande computazionali riguardanti il nucleo riguardano il suo essere vuoto o meno e il fatto che una particolare distribuzione di payoff ne faccia parte.

La definizione di nucleo conduce naturalmente a un sistema di disequazioni lineari, come mostrato di seguito (le incognite sono le variabili x_1, \dots, x_n , mentre i valori $v(C)$ sono costanti):

$$\begin{aligned}x_i &\geq v(\{i\}) \quad \text{per ogni } i \in N \\ \sum_{i \in N} x_i &= v(N) \\ \sum_{i \in C} x_i &\geq v(C) \quad \text{per ogni } C \subseteq N\end{aligned}$$

Ogni soluzione di questo sistema definirà un'imputazione appartenente al nucleo. Possiamo formulare le diseguaglianze come un programma lineare utilizzando una funzione obiettivo fittizia (per esempio, massimizzando $\sum_{i \in N} x_i$), il che ci consente di calcolare le imputazioni in tempo polinomiale nel numero di disequazioni. La difficoltà è che ciò genera un numero esponenziale di disequazioni (una per ognuna delle 2^n coalizioni possibili). Perciò, questo approccio fornisce un algoritmo per verificare che il nucleo sia non vuoto eseguibile in tempo esponenziale. La possibilità di fare di meglio dipende dal gioco studiato: per molti classi di giochi cooperativi, il problema di verificare che il nucleo non è vuoto è co-NP-completo. Vedremo un esempio più avanti.

Prima di proseguire, vediamo un esempio di gioco superadditivo con nucleo vuoto. Il gioco ha tre giocatori $N = \{1, 2, 3\}$ e una funzione caratteristica definita come segue:

$$v(C) = \begin{cases} 1 & \text{se } |C| \geq 2 \\ 0 & \text{altrimenti.} \end{cases}$$

Consideriamo ora un'imputazione (x_1, x_2, x_3) per questo gioco. Poiché $v(N) = 1$, allora almeno un giocatore i ha $x_i > 0$ mentre gli altri due ottengono complessivamente un payoff minore di 1. Per questi due giocatori sarebbe vantaggioso formare una coalizione senza il giocatore i e spartire tra loro il valore 1. Ma poiché questo vale per tutte le imputazioni, il nucleo deve essere vuoto.

Il nucleo formalizza l'idea che la grande coalizione sia *stabile*, nel senso che nessuna coalizione trae vantaggio dal separarsi da essa. Tuttavia il nucleo potrebbe contenere imputazioni *irragionevoli*, ovvero percepite come inique da uno o più giocatori. Supponiamo che $N = \{1, 2\}$; e che la funzione caratteristica v sia definita come segue:

$$\begin{aligned}v(\{1\}) &= v(\{2\}) = 5 \\v(\{1, 2\}) &= 20.\end{aligned}$$

In questo caso la cooperazione genera un surplus di 10 rispetto a ciò che i giocatori potrebbero ottenere lavorando da soli, quindi in questo scenario ha senso cooperare. Ora, si vede facilmente che l'imputazione $(6, 14)$ fa parte del nucleo di questo gioco: nessun giocatore ottiene un'utilità maggiore scostandosi da essa. Dal punto di vista del giocatore 1, però, questa imputazione può apparire irragionevole perché assegna $9/10$ del surplus al giocatore 2. La nozione di nucleo permette quindi di stabilire quando è possibile formare una grande coalizione, ma non indica come distribuire il payoff.

valore di Shapley

Il valore di Shapley è un'elegante proposta per come suddividere il valore $v(N)$ tra i giocatori, posto che si sia formata la grande coalizione N . Formulato dal premio Nobel Lloyd Shapley nei primi anni 1950, il valore di Shapley rappresenta uno schema di distribuzione *equo*.

Che cosa significa *equo*? Sarebbe ingiustificato distribuire $v(N)$ sulla base del colore degli occhi, del genere o del colore della pelle dei giocatori. Gli studenti suggeriscono spesso che $v(N)$ dovrebbe essere suddiviso in parti uguali, il che sembra corretto finché non si considera che in questo modo si assegnerebbe lo stesso compenso ai giocatori che hanno contribuito molto e a quelli che non hanno contribuito affatto. L'idea di Shapley è che l'unico modo equo per suddividere il valore $v(N)$ sia quello di farlo in base a quanto ciascun giocatore ha *contribuito* alla creazione del valore $v(N)$.

contributo marginale

Anzitutto occorre definire la nozione di **contributo marginale** di un giocatore. Il contributo marginale del giocatore i per la coalizione C è il valore che i aggiungerebbe (o toglierebbe) unendosi alla coalizione C . Formalmente, il contributo marginale apportato da i alla coalizione C è indicato da $mc_i(C)$ (dall'inglese *marginal contribution*):

$$mc_i(C) = v(C \cup \{i\}) - v(C).$$

Un primo tentativo di definire uno schema di suddivisione del payoff in linea con il suggerimento di Shapley che i giocatori debbano essere ricompensati in base al loro contributo consiste nell'assegnare a ciascun giocatore i il valore che egli aggiungerebbe alla coalizione contenente tutti gli altri giocatori:

$$mc_i(N - \{i\}).$$

Il problema è che ciò implicitamente assume che il giocatore i sia *l'ultimo* a entrare nella coalizione. Shapley suggerì quindi di considerare tutti i possibili modi in cui la grande coalizione può formarsi, ovvero tutti i possibili ordinamenti dei giocatori N , e il valore che i aggiunge rispetto ai giocatori che lo precedono nell'ordinamento. Allora, un giocatore i dovrebbe essere ricompensato con il *contributo marginale medio*, su tutti i possibili ordinamenti dei giocatori, apportato da i all'insieme di giocatori che lo precedono nell'ordinamento.

Poniamo che \mathcal{P} indichi tutte le possibili permutazioni (ordinamenti) dei giocatori N , e indichiamo gli elementi di \mathcal{P} con p, p', \dots e così via. Dati $p \in \mathcal{P}$ e $i \in N$, indichiamo con p_i l'in-

sieme dei giocatori che precedono i nell'ordinamento p . Allora il valore di Shapley per un gioco G è l'imputazione $\phi(G) = (\phi_1(G), \dots, \phi_n(G))$ definita come segue:

$$\phi_i(G) = \frac{1}{n!} \sum_{p \in \mathcal{P}} mc_i(p_i). \quad (18.1)$$

Ciò dovrebbe bastare per convincerci che il valore di Shapley è una proposta ragionevole. Ma il fatto notevole è che si tratta dell'*unica* soluzione che soddisfa l'insieme di assiomi che definiscono uno schema “equo” di distribuzione dei payoff.

Definiamo **giocatore fittizio (dummy)** un giocatore i che non aggiunge mai valore a una coalizione, ovvero: $mc_i(C) = 0$ per ogni $C \subseteq N - \{i\}$. Diremo che due giocatori i e j sono **giocatori simmetrici** se apportano sempre contributi *identici* alle coalizioni, ovvero: $mc_i(C) = mc_j(C)$ per ogni $C \subseteq N - \{i, j\}$. Infine, se $G = (N, v)$ e $G' = (N, v')$ sono giochi con lo stesso insieme di giocatori, il gioco $G + G'$ è il gioco che ha lo stesso insieme di giocatori e una funzione caratteristica v'' definita da $v''(C) = v(C) + v'(C)$.

Date queste definizioni, possiamo definire gli assiomi di equità soddisfatti dal valore di Shapley:

- **Efficienza:** $\sum_{i \in N} \phi_i(G) = v(N)$ (tutto il valore deve essere distribuito).
- **Giocatore fittizio:** se i è un giocatore fittizio in G , allora $\phi_i(G) = 0$ (i giocatori che non contribuiscono non devono ricevere nulla)
- **Simmetria:** se i e j sono simmetrici in G , allora $\phi_i(G) = \phi_j(G)$ (i giocatori che apportano contributi identici devono ricevere payoff identici).
- **Additività:** il valore è additivo tra i giochi: per ogni coppia di giochi $G = (N, v)$ e $G' = (N, v')$, e per ogni giocatore $i \in N$, abbiamo $\phi_i(G + G') = \phi_i(G) + \phi_i(G')$.

L'assioma dell'additività è in effetti piuttosto tecnico. Tuttavia, se lo accettiamo come requisito possiamo stabilire la seguente fondamentale proprietà: *il valore di Shapley è l'unico modo per distribuire il valore della coalizione soddisfacendo questi assiomi di equità.*



18.3.3 Computazioni nei giochi cooperativi

Dal punto di vista teorico, ora abbiamo una soluzione soddisfacente. Ma dal punto di vista computazionale dobbiamo stabilire come *rappresentare sinteticamente* i giochi cooperativi e come *calcolare in modo efficiente* i concetti di soluzione, quali il nucleo e il valore di Shapley.

La rappresentazione più ovvia di una funzione caratteristica sarebbe una tabella che elenchi i valori $v(C)$ per tutte le 2^n coalizioni: impraticabile per valori grandi di n . Sono stati sviluppati vari approcci per rappresentare in modo compatto i giochi cooperativi, che si possono distinguere in base alla loro *completezza*. Uno schema di rappresentazione completo è uno schema in grado di rappresentare *qualsiasi* gioco cooperativo. L'inconveniente degli schemi di rappresentazione completi è che ci saranno sempre alcuni giochi che non potranno essere rappresentati in modo compatto. Un'alternativa consiste nell'utilizzare uno schema di rappresentazione che garantisca la compattezza, seppur non la completezza.

Reti di contributi marginali

Descriviamo ora uno schema di rappresentazione chiamato **reti di contributi marginali** (o reti MC). Ne utilizzeremo una versione leggermente semplificata per facilitare la presentazione. Tale semplificazione lo rende incompleto; la versione originale dello schema MC-net, dall'inglese *marginal contribution net*) è invece una rappresentazione completa.

rete di contributi marginali

L'idea alla base delle reti di contributi marginali è quella di rappresentare la funzione caratteristica di un gioco (N, v) come un insieme di regole coalizione-valore per definire il valore della coalizione; le regole hanno la forma (C_i, x_i) , dove $C_i \subseteq N$ è una coalizione e x_i è un numero. Per calcolare il valore di una coalizione C , si sommano semplicemente i valori

di tutte le regole (C_i, x_i) tali che $C_i \subseteq C$. Quindi, dato un insieme di regole $R = \{(C_1, x_1), \dots, (C_k, x_k)\}$, la corrispondente funzione caratteristica è:

$$\nu(C) = \sum \{x_i \mid (C_i, x_i) \in R \text{ e } C_i \subseteq C\}.$$

Supponiamo di avere un insieme R contenente le tre regole seguenti:

$$\{\{1, 2\}, 5\}, \quad \{\{2\}, 2\}, \quad \{\{3\}, 4\}.$$

Allora abbiamo, per esempio:

$$\nu(\{1\}) = 0 \text{ (non ci sono regole per questo caso)},$$

$$\nu(\{3\}) = 4 \text{ (terza regola)},$$

$$\nu(\{1, 3\}) = 4 \text{ (terza regola)},$$

$$\nu(\{2, 3\}) = 6 \text{ (seconda e terza regola)}, \text{ e}$$

$$\nu(\{1, 2, 3\}) = 11 \text{ (prima, seconda e terza regola)}.$$

Con questa rappresentazione possiamo calcolare il valore di Shapley in tempo polinomiale. L'idea chiave è che ogni regola può essere intesa come la definizione di un gioco a parte, in cui i giocatori sono simmetrici. Richiamando gli assiomi di additività e di simmetria di Shapley, quindi, il valore di Shapley $\phi_i(R)$ del giocatore i nel gioco associato all'insieme di regole R è semplicemente:

$$\phi_i(R) = \sum_{(C, x) \in R} \begin{cases} \frac{x}{|C|} & \text{se } i \in C \\ 0 & \text{altrimenti.} \end{cases}$$

Questa versione delle reti di contributi marginali non è uno schema di rappresentazione *completo*: esistono giochi la cui funzione caratteristica non può essere rappresentata utilizzando insiemi di regole della forma sopra descritta. Un tipo più ricco di rete di contributi marginali ammette anche regole della forma (ϕ, x) , dove ϕ è una formula della logica proposizionale sui giocatori N : una coalizione C soddisfa la condizione ϕ se corrisponde a un assegnamento che soddisfa ϕ .

Questo apre una rappresentazione completa: nel caso peggiore, ci occorre una regola per ogni possibile coalizione. Inoltre, con questo schema il valore di Shapley può essere calcolato in tempo polinomiale; la procedura è più complicata rispetto al caso con le semplici regole che abbiamo descritto, anche se il principio base è lo stesso; si vedano i riferimenti nelle note storiche e bibliografiche al termine del capitolo.

Strutture di coalizioni per il massimo benessere sociale

Si apre a una prospettiva diversa sui giochi cooperativi se si assume che gli agenti abbiano uno scopo comune. Per esempio, se pensiamo agli agenti come ai lavoratori di un'azienda, allora le considerazioni strategiche riguardanti la formazione della coalizione, a cui si riferisce il nucleo, non sono rilevanti. Potremmo invece voler organizzare la forza lavoro (gli agenti) in squadre, per massimizzare la produttività complessiva. Più in generale, il compito è trovare una coalizione che massimizzi il *benessere sociale* (*social welfare*) del sistema, definito come la somma dei valori delle singole coalizioni. Indichiamo il benessere sociale di una struttura di coalizioni CS con $sw(CS)$, e lo definiamo come segue:

$$sw(CS) = \sum_{C \in CS} \nu(C).$$

Allora una struttura di coalizioni CS^* socialmente ottima rispetto a G massimizza questa quantità. Trovare una struttura di coalizioni socialmente ottima è un problema di calcolo molto naturale, studiato anche al di fuori dell'ambito dei sistemi multiagente: viene a volte

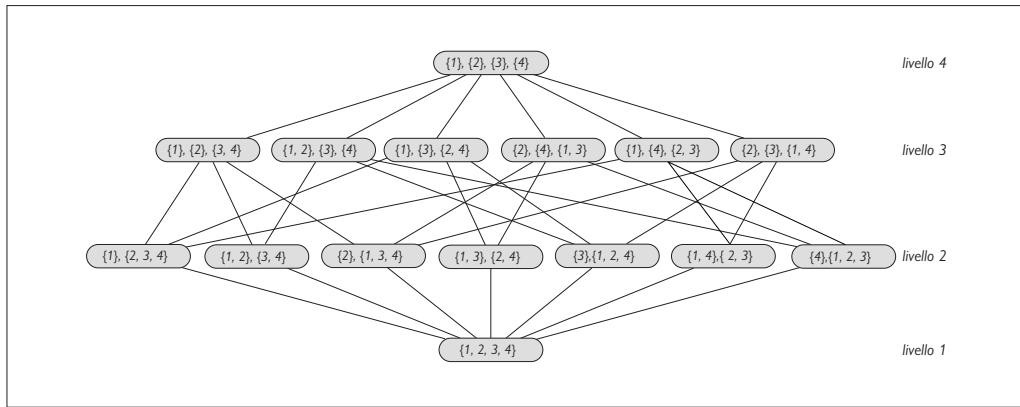


Figura 18.7 Grafo delle strutture di coalizioni per $N = \{1, 2, 3, 4\}$. Al livello 1 si trovano le strutture di coalizioni che contengono una sola coalizione; al livello 2 le strutture di coalizioni che contengono due coalizioni, e così via.

chiamato **problema di partizione di un insieme**. Sfortunatamente, il problema è NP-difficile, perché il numero di possibili strutture di coalizioni cresce esponenzialmente con il numero di giocatori.

Trovare la struttura di coalizioni ottima attraverso una semplice ricerca esaustiva, quindi, è generalmente impossibile. Un approccio influente alla formazione di strutture di coalizioni ottime è quello basato sull'idea di ricercare un sottospazio del **grafo delle strutture di coalizioni**. Per spiegare questa idea ricorriamo a un esempio.

Supponiamo di avere un gioco con quattro agenti, $N = \{1, 2, 3, 4\}$. Per questo insieme di agenti esistono quindici possibili strutture di coalizioni. Possiamo organizzarle in un grafo come quello della Figura 18.7, in cui i nodi al livello ℓ corrispondono a tutte le strutture di coalizioni con esattamente ℓ coalizioni. Un passaggio verso l'alto nel grafo rappresenta la divisione di una coalizione del nodo inferiore in due coalizioni separate nel nodo superiore.

Per esempio, c'è un collegamento tra $\{\{1\}, \{2, 3, 4\}\}$ e $\{\{1\}, \{2\}, \{3, 4\}\}$ perché quest'ultima struttura di coalizioni si ottiene dalla precedente scomponendo la coalizione $\{2, 3, 4\}$ nelle coalizioni $\{2\}$ e $\{3, 4\}$.

La struttura di coalizioni ottima CS^* si trova da qualche parte all'interno del grafo, e sembra dunque che per trovarla sia necessario valutare ogni singolo nodo di quest'ultimo. Consideriamo però le due righe inferiori del grafo, i livelli 1 e 2. Ogni possibile coalizione (a eccezione di quella vuota) è presente in questi due livelli (ovviamente non è presente ogni possibile struttura di coalizioni). Supponiamo ora di restringere la ricerca di una possibile struttura di coalizioni a questi due soli livelli: non andremo più in alto nel grafo. Sia CS' la migliore struttura di coalizioni individuata in questi due livelli e sia CS^* la migliore in generale. Poniamo che C^* sia una delle coalizioni con il valore più alto tra tutte le possibili coalizioni:

$$C^* \in \arg \max_{C \subseteq N} v(C).$$

Il valore della migliore struttura di coalizioni che troviamo nei primi due livelli del grafo deve essere almeno pari al valore della migliore coalizione: $sw(CS') \geq v(C^*)$. Ciò perché nei primi due livelli del grafo appare ogni coalizione possibile, in almeno una struttura di coalizioni. Ipotizziamo dunque il caso peggiore, ovvero che $sw(CS') = v(C^*)$.

Confrontiamo il valore di $sw(CS')$ con quello di $sw(CS^*)$. Poiché $sw(CS^*)$ è il più alto tra i valori delle possibili strutture di coalizioni, e poiché ci sono n agenti ($n = 4$ nel caso della Figura 18.7), allora il più alto valore possibile di $sw(CS^*)$ sarà $nv(C^*) = n \cdot sw(CS^*)$. In altre parole, nel caso peggiore il valore della migliore struttura di coalizioni che troviamo nei pri-

**problema
di partizione
di un insieme**

**grafo delle strutture
di coalizioni**

mi due livelli del grafo sarà $\frac{1}{n}$ del valore della migliore in assoluto, dove n è il numero di agenti. Quindi, sebbene la ricerca nei primi due livelli del grafo non garantisca di individuare la struttura di coalizioni *ottima*, essa *garantisce* di individuarne una non peggiore di $\frac{1}{n}$ di quella ottima. Nella pratica, sarà spesso molto migliore di così.

18.4 Decisioni collettive

Passiamo ora dalla progettazione di agenti alla **progettazione di meccanismi**, ovvero al problema di progettare il gioco giusto a cui far giocare un insieme di agenti. Formalmente, un **meccanismo** consiste dei seguenti elementi.

1. Un linguaggio per descrivere l’insieme delle strategie ammissibili che gli agenti possono adottare.
2. Un agente speciale, detto **centro**, che raccoglie le informazioni sulle scelte strategiche degli agenti partecipanti al gioco (per esempio, il banditore è il centro di un’asta).
3. Una regola che determina i risultati, nota a tutti gli agenti, che il centro utilizza per assegnare i payoff a ogni agente in base alle loro scelte strategiche.

Questo paragrafo esamina alcuni dei meccanismi più importanti.

18.4.1 Assegnare compiti con il contract net protocol

Il **contract net protocol** (o **protocollo di rete contrattuale**) è probabilmente la più antica e più importante tecnica di soluzione di problemi multiagente studiata nel campo dell’IA. È un protocollo di alto livello per l’assegnazione di compiti. Come suggerito dal nome, la rete contrattuale si ispira al modo in cui i contratti vengono utilizzati dalle aziende.

Il contract net protocol ha quattro fasi principali, illustrate nella Figura 18.8. Il processo inizia con un agente che identifica la necessità di un’azione cooperativa rispetto a un determinato compito. La necessità può presentarsi perché l’agente non ha la capacità di svolgere l’attività da solo o perché una soluzione cooperativa potrebbe essere migliore sotto qualche aspetto (più rapida, più efficiente, più accurata).

L’agente segnala il compito agli altri agenti della rete con un messaggio di **annuncio del compito**, quindi agisce come **amministratore** (*manager*) dell’esecuzione di tale compito per tutta la sua durata. Il messaggio di annuncio deve contenere informazioni sufficienti ai destinatari per valutare se sono interessati e in grado di fare un’offerta rispetto al compito. Le specifiche informazioni contenute nel messaggio dipenderanno dalla specifica area applicativa. Potrebbe trattarsi di codice da eseguire, oppure della specifica logica di un obiettivo da raggiungere. L’annuncio può anche contenere altre informazioni eventualmente necessarie ai destinatari, come scadenze, requisiti di qualità e così via.

Quando un agente riceve un annuncio riguardo a un compito, deve valutarla rispetto alle proprie capacità e preferenze. In particolare, ogni agente deve determinare se abbia la capacità di eseguire il compito e, in secondo luogo, se desideri eseguirlo. Su queste basi, può poi decidere di presentare un’**offerta** per il compito. L’offerta indicherà tipicamente le capacità dell’agente rispetto al compito annunciato e tutte le clausole e le condizioni che ne regoleranno l’esecuzione.

In generale, è possibile che un amministratore riceva molte offerte in risposta a un singolo annuncio. In base alle informazioni contenute nelle offerte, l’amministratore seleziona l’agente più adeguato (o gli agenti) all’esecuzione del compito. Gli agenti selezionati vengono informati tramite un messaggio di aggiudicazione e prendono in appalto il compito, assumendone la responsabilità fino al suo completamento.

Le principali operazioni di calcolo necessarie per implementare il contract net protocol possono essere riepilogate come segue.

centro

contract net protocol

annuncio del compito amministratore

offerta

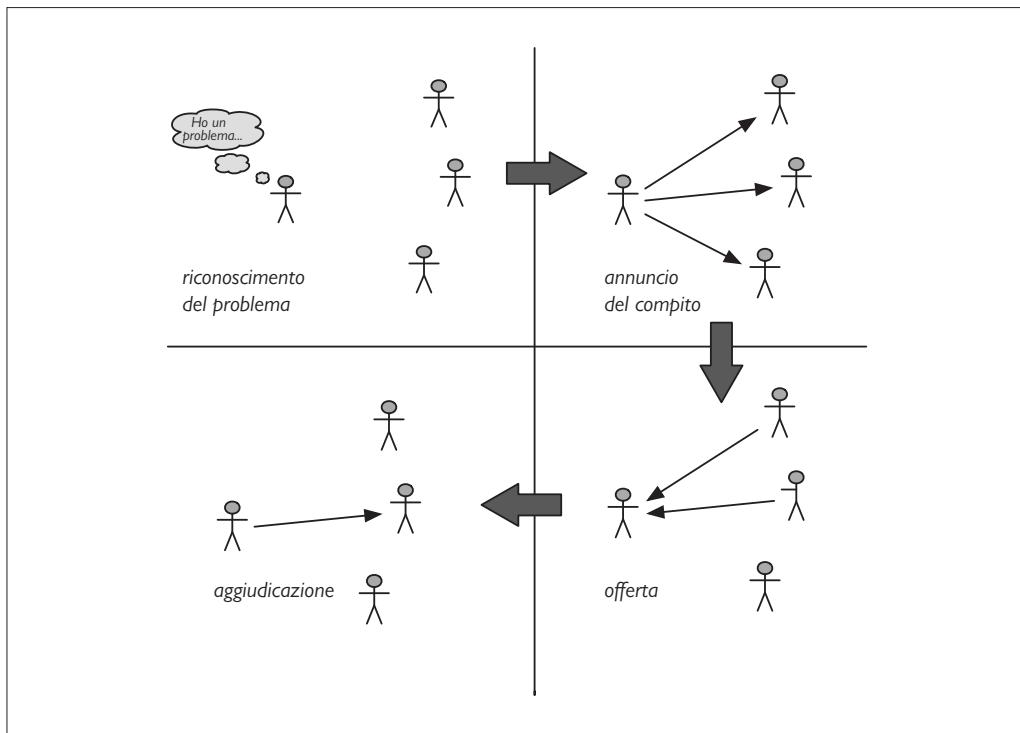


Figura 18.8
Il contract net protocol per l'assegnazione di compiti.

- *Elaborazione dell'annuncio del compito.* Ricevendo un annuncio, l'agente decide se desidera fare un'offerta per il compito annunciato.
- *Elaborazione delle offerte.* Ricevendo più offerte, l'amministratore deve decidere a quale agente assegnare il compito, e poi procedere all'aggiudicazione.
- *Elaborazione dell'aggiudicazione.* Gli offerenti selezionati (appaltatori) devono tentare di eseguire il compito, che potrebbe implicare la creazione di nuovi compiti secondari, che verranno segnalati con ulteriori annunci.

Nonostante la sua semplicità (o forse grazie a essa), il contract net protocol è probabilmente l'approccio alla risoluzione di problemi cooperativi più utilizzato e più studiato. È naturalmente applicabile a molti scenari: una sua variante, per esempio, entra in azione ogni volta che si richiede un'automobile con Uber.

18.4.2 Allocare risorse scarse con le aste

Uno dei problemi più importanti nei sistemi multiagente è quello dell'allocazione delle risorse scarse; ma potremmo limitarci a dire “allocazione delle risorse”, dato che in pratica la maggior parte delle risorse utili può essere considerata scarsa in qualche senso. L'**asta** è il meccanismo di allocazione di risorse più importante. La situazione più semplice in un'asta è che vi sia una singola risorsa e più possibili **offerenti**. Ogni offerente i associa all'oggetto all'asta un valore di utilità v_i .

In alcuni casi, ogni offerente assegna all'oggetto un diverso **valore privato**. Per esempio, un maglione fuori moda potrebbe essere interessante per un offerente e senza valore per un altro.

In altri casi, come nelle aste dei diritti di esplorazione petrolifera, l'oggetto ha un **valore comune**: il giacimento produrrà una certa somma di denaro, X , e tutti gli offerenti valutano il denaro allo stesso modo (ma esiste incertezza sul valore effettivo di X). I diversi offerenti

asta

offerenti

asta al rialzo
asta inglese

efficiente

collusione

strategia dominante

hanno informazioni differenti e, di conseguenza, differenti stime del vero valore dell’oggetto. In ogni caso, ciascun offerente avrà il proprio v_i e la possibilità, al momento o nei momenti opportuni durante l’asta, di effettuare un’offerta b_i . L’offerta più alta, b_{max} , si aggiudica l’oggetto ma il prezzo pagato non è necessariamente b_{max} bensì dipende da come è strutturato il meccanismo.

Il meccanismo di asta più noto è l'**asta al rialzo**, o **asta inglese**, in cui il centro inizia chiedendo un’offerta minima (o **riserva**) b_{min} . Se un offerente è disponibile a pagare tale importo, il centro chiede allora $b_{min} + d$, per un determinato incremento d , per poi proseguire al rialzo. L’asta termina quando nessuno è disposto a fare nuove offerte. L’ultimo offerente si aggiudica allora l’oggetto, pagando la somma offerta.

Come stabilire se si tratta di un buon meccanismo? Un obiettivo è massimizzare il profitto atteso del venditore; un altro è massimizzare una forma di utilità globale. Questi obiettivi sono in una certa misura sovrapposti, perché uno degli aspetti della massimizzazione dell’utilità globale è fare in modo che il vincitore dell’asta sia l’agente che valuta maggiormente l’oggetto (e che quindi è disposto a pagarla di più). Diciamo che un’asta è **efficiente** quando i beni vanno all’agente che li valuta di più. L’asta al rialzo di solito è efficiente oltre a massimizzare il profitto. Se però il prezzo di riserva è troppo elevato, l’offerente che valuta maggiormente l’oggetto potrebbe non fare offerte; se invece la riserva è troppo bassa, il venditore potrebbe guadagnare meno.

Probabilmente, le cose più importanti che un meccanismo di asta può fare sono incoraggiare la partecipazione di un numero sufficiente di offerenti e scoraggiare la **collusione**. La collusione è un accordo disonesto o illegale tra due o più offerenti per manipolare i prezzi. Può nascere da patti segreti oppure tacitamente, entro le regole del meccanismo. Per esempio, nel 1999 in Germania vennero messi in vendita dieci blocchi di frequenze per la telefonia cellulare con un’asta simultanea (venivano accettate offerte su tutti e dieci i blocchi contemporaneamente), con la regola che ogni offerta su un determinato blocco doveva rappresentare un incremento di almeno il 10% rispetto all’offerta precedente. Si presentarono solamente due offerenti credibili e il primo, Mannesman, offrì 20 milioni di marchi per i blocchi 1-5 e 18,8 milioni per i blocchi 6-10. Perché 18,8? Uno dei dirigenti di T-Mobile disse che la sua azienda aveva “interpretato le offerte di Mannesman come una proposta”. Entrambe le parti sapevano calcolare che un incremento del 10% su 18,8 milioni porta a 19,99 milioni; le offerte di Mannesman vennero quindi interpretate come un messaggio: “ognuno di noi può aggiudicarsi metà dei blocchi per 20 milioni; non roviniamo tutto facendo salire le offerte”. Di fatto, T-Mobile offrì 20 milioni per i blocchi 6-10 e non ci furono ulteriori offerte.

Il governo tedesco ottenne meno di quanto si aspettava, perché i due concorrenti erano riusciti a utilizzare il meccanismo di offerta per giungere tacitamente all’accordo di non competere. Dal punto di vista del governo, si sarebbe potuto ottenere un risultato migliore con una di queste variazioni del meccanismo: un prezzo di riserva più alto; una gara a offerte segrete, in modo che i concorrenti non potessero comunicare tramite le offerte; incentivi al coinvolgimento di un terzo offerente. La regola del 10% forse fu un errore nella progettazione del meccanismo, perché facilitò una precisa segnalazione di Mannesman a T-Mobile.

In generale, la presenza di più offerenti è un fatto positivo tanto per il venditore quanto per la funzione di utilità globale, anche se l’utilità globale potrebbe risentire del costo del tempo sprecato dagli offerenti che non hanno possibilità di vittoria. Un modo per attirare più offerenti consiste nel rendere il meccanismo più semplice per loro: se costretti a fare troppa ricerca o troppi calcoli, i potenziali offerenti potrebbero decidere di portare il loro denaro altrove.

Quindi è desiderabile che gli offerenti abbiano una **strategia dominante**. Ricordiamo che “dominante” significa che la strategia è preferibile a tutte le altre, il che significa a sua volta che un agente può adottarla senza ulteriori considerazioni. Chi ha una strategia dominante può semplicemente fare un’offerta senza perdere tempo a considerare le possibili strategie

altrui. Un meccanismo che dà agli agenti una strategia dominante è detto meccanismo **a prova di strategia** (*strategy-proof*). Se, come solitamente accade, la strategia implica che gli offerenti rivelino il loro vero valore, v_i , allora si parla di asta **rivelatrice** (*truth-revealing* o *truthful*); si utilizza anche l'espressione **compatibile con gli incentivi**. Il **principio di rivelazione** afferma che ogni meccanismo può essere trasformato in un equivalente meccanismo rivelatore, quindi una parte del lavoro di progettazione consiste nel trovare questi equivalenti.

L'asta al rialzo gode della maggior parte delle proprietà desiderabili. L'offerente con il più alto valore v_i si aggiudica i beni al prezzo $b_o + d$, dove b_o è l'offerta più alta tra quelle degli altri agenti e d è l'incremento del banditore.³ Gli offerenti hanno una semplice strategia dominante: continuare a fare offerte finché il prezzo attuale è inferiore al proprio v_i . Il meccanismo non è molto rivelatore, perché l'offerente vincente svela solo che il suo $v_i \geq b_o + d$; abbiamo un limite inferiore per v_i ma non il valore esatto.

Uno svantaggio (dal punto di vista del venditore) dell'asta al rialzo è che può scoraggiare la concorrenza. Supponiamo che a un'asta di frequenze telefoniche partecipi un'azienda avvantaggiata, che tutti sanno avere clientela e infrastrutture già esistenti e quindi la possibilità di realizzare profitti maggiori rispetto agli altri. I potenziali concorrenti capiscono di non avere possibilità in un'asta a offerte crescenti, perché l'azienda avvantaggiata può sempre fare un'offerta più alta. Quindi i concorrenti potrebbero non partecipare affatto e l'azienda avvantaggiata finirebbe per vincere al prezzo di riserva.

Un'altra proprietà negativa dell'asta inglese è data dai suoi elevati costi di comunicazione. L'asta ha luogo in una sala, oppure tutti gli offerenti devono disporre di linee di comunicazione ad alta velocità e sicure; in entrambi i casi devono avere tempo da dedicare a diverse tornate di offerte.

Un meccanismo alternativo, che richiede molta meno comunicazione, è l'**asta a offerta segreta** (o a busta chiusa, *sealed-bid auction*). Ogni partecipante fa una singola offerta e la comunica al banditore, senza che gli altri la vedano. Con questo meccanismo, non c'è più una semplice strategia dominante. Se il nostro valore è v_i e riteniamo che tutti gli altri agenti offriranno al massimo b_o , allora dovremmo offrire $b_o + \varepsilon$, per un piccolo valore di ε , se tale somma è minore di v_i . Perciò la nostra offerta dipende dalla nostra stima delle offerte degli altri agenti e ciò richiede del lavoro aggiuntivo. Inoltre, va notato che l'agente con il valore v_i più alto potrebbe non vincere l'asta. In compenso, l'asta è più competitiva e riduce lo squilibrio a favore degli offerenti avvantaggiati.

Una piccola modifica del meccanismo dell'asta a offerta segreta è quella che conduce all'**asta al secondo prezzo a offerta segreta**, nota anche come **asta di Vickrey**.⁴ In queste aste il vincitore paga il prezzo della *seconda* offerta più alta, b_o , invece dell'importo della propria offerta. Questa semplice modifica elimina completamente i complessi ragionamenti delle normali aste a offerta segreta (o **al primo prezzo**), perché la strategia dominante ora è semplicemente offrire v_i ; il meccanismo è rivelatore. Notiamo che l'utilità dell'agente i espressa in termini della sua offerta b_i , del suo valore v_i e della migliore offerta degli altri agenti, b_o , è

$$U_i = \begin{cases} (v_i - b_o) & \text{se } b_i > b_o \\ 0 & \text{altrimenti.} \end{cases}$$

Per verificare che $b_i = v_i$ è una strategia dominante, notiamo che quando $(v_i - b_o)$ è positivo, ogni offerta che si aggiudichi l'asta è ottima, e in particolare l'offerta v_i è vincente. Quando invece $(v_i - b_o)$ è negativo, è ottima ogni offerta perdente e in particolare v_i è perdente. Quindi offrire v_i è ottimo per tutti i possibili valori di b_o e in effetti v_i è l'unica offerta che abbia

a prova di strategia

principio di rivelazione

asta a offerta segreta

asta al secondo prezzo a offerta segreta
asta di Vickrey

³ Esiste in realtà una piccola possibilità che l'agente con il v_i più alto non riesca ad aggiudicarsi il bene, nel caso in cui $b_o < v_i < b_o + d$. La probabilità che questo accada può essere resa arbitrariamente piccola riducendo l'incremento d .

⁴ Così chiamata in riferimento a William Vickrey (1914–1996), che nel 1996 vinse il premio Nobel per l'economia grazie a questo lavoro e che morì per un attacco cardiaco tre giorni più tardi.

**teorema
dell'equivalenza
dei ricavi**

questa proprietà. Per la sua semplicità e per i minimi requisiti di calcolo tanto per il venditore che per gli offerenti, l'asta di Vickrey è ampiamente utilizzata nei sistemi di IA distribuiti.

I motori di ricerca su Internet effettuano diverse migliaia di aste ogni anno per vendere la pubblicità mostrata con i risultati delle ricerche e i siti di aste online gestiscono beni per 100 miliardi di dollari, utilizzando varianti dell'asta di Vickrey. Va notato che il valore atteso per il venditore è b_o , che equivale al valore atteso dell'asta inglese quando l'incremento d tende a zero. Si tratta di un risultato molto generale: il **teorema dell'equivalenza dei ricavi** afferma che, poste alcune condizioni precauzionali, ogni meccanismo di asta per il quale gli offerenti hanno valori v_i noti solo a loro stessi (ma conoscono la distribuzione di probabilità da cui questi valori sono tratti) genera il medesimo ricavo atteso. Questo principio significa che i vari meccanismi non si differenziano in base alla generazione di ricavi, bensì rispetto ad altre qualità.

Sebbene l'asta al secondo prezzo abbia un meccanismo rivelatore, risulta che un'asta di n beni al $(n + 1)$ -esimo prezzo non è rivelatrice. Molti motori di ricerca Internet utilizzano un meccanismo con il quale mettono all'asta n spazi pubblicitari su una pagina. Il miglior offerente si aggiudica lo spazio migliore, il secondo miglior offerente il secondo spazio e così via. Ogni vincitore paga l'importo dell'offerta immediatamente inferiore, con l'intesa che il pagamento viene effettuato solamente se l'utente del motore di ricerca fa clic sull'inserzione. Gli spazi più in alto sono considerati di maggior valore perché è più probabile che un utente li noti e faccia clic su di essi.

Immaginiamo che tre offerenti, b_1 , b_2 e b_3 , valutino un clic $v_1 = 200$, $v_2 = 180$ e $v_3 = 100$, che siano disponibili $n = 2$ spazi e che sia noto che gli utenti fanno clic sul primo spazio nel 5% dei casi e sul secondo nel 2%. Se tutti i partecipanti offrono in modo fedele alle proprie preferenze, allora b_1 ottiene il primo spazio e paga 180, per un rendimento atteso di $(200 - 180) \times 0,05 = 1$. Il secondo spazio va a b_2 . Ma b_1 vede che, offrendo una cifra compresa tra 101 e 179, concederebbe il primo spazio a b_2 , ottenendo il secondo spazio e un rendimento atteso di $(200 - 100) \times 0,02 = 2$. Quindi b_1 può in questo caso raddoppiare il proprio rendimento atteso offrendo meno di quanto realmente valuti il bene.

In generale, gli offerenti di un'asta al $(n + 1)$ -esimo prezzo devono dedicare molte energie all'analisi delle offerte altrui per determinare la strategia migliore per sé; non esiste una semplice strategia dominante.

Aggarwal *et al.* (2006) mostrano che esiste un unico meccanismo rivelatore per questo problema dell'assegnazione di più spazi, nel quale chi si aggiudica lo spazio j paga il prezzo corrispondente solo per i clic aggiuntivi che sono disponibili nello spazio j e non nello spazio $j + 1$; per i clic rimanenti, paga il prezzo dello spazio inferiore. Nel nostro esempio, b_1 offrirebbe 200 secondo le proprie preferenze e pagherebbe 180 per i 0,05 – 0,02 = 0,03 clic aggiuntivi sul primo spazio, mentre per i restanti 0,02 clic pagherebbe solamente il prezzo dello spazio inferiore, pari a 100. Quindi, il rendimento totale per b_1 sarebbe $(200 - 180) \times 0,03 + (200 - 100) \times 0,02 = 2,6$.

Un altro esempio del modo in cui le aste possono entrare in gioco nell'IA è quello di un insieme di agenti che devono decidere se cooperare in un piano congiunto. Hunsberger e Grosz (2000) mostrano che un metodo efficiente è un'asta in cui gli agenti fanno offerte per i vari ruoli del piano congiunto.

Beni comuni

Consideriamo ora un altro tipo di gioco, in cui ogni nazione stabilisce le proprie politiche per limitare l'inquinamento dell'aria. Ogni nazione può scegliere: può ridurre l'inquinamento, al costo di -10 punti per la realizzazione dei cambiamenti necessari, oppure continuare a inquinare, ottenendo l'utilità netta -5 (per le spese sanitarie aggiuntive e così via) e contribuendo -1 punto a tutte le altre nazioni (perché l'aria è condivisa da tutti). Chiaramente, la strategia dominante per ciascuna nazione è "continuare a inquinare". Se però le nazioni

sono 100 e tutte adottano questa politica, allora ogni nazione ha un'utilità complessiva pari a -104, mentre se tutte le nazioni riducessero l'inquinamento, per ognuna l'utilità sarebbe -10. Questa situazione è detta **tragedia dei beni comuni**: se nessuno deve pagare per l'utilizzo di una risorsa comune, allora questa può essere sfruttata in un modo che conduce a una minore utilità per tutti gli agenti. È simile al dilemma del prigioniero: esiste un'altra soluzione del gioco, migliore per tutte le parti, ma sembra non esistere un modo in cui degli agenti razionali possano arrivare a tale soluzione.

Uno degli approcci alla soluzione di questo problema consiste nel modificare il meccanismo facendo pagare a ogni agente l'utilizzo delle risorse collettive. Più in generale, occorre rendere esplicite tutte le **esternalità**, ovvero gli effetti sull'utilità globale, che non vengono riconosciuti nelle transazioni dei singoli agenti.

La parte difficile è stabilire correttamente i prezzi. Al limite, questo approccio porta alla creazione di un meccanismo in cui ogni agente, pur potendo effettuare solo decisioni locali, viene efficacemente chiamato a massimizzare l'utilità globale. Per questo esempio, una tassa sulle emissioni di diossido di carbonio è un esempio di meccanismo che rende costoso utilizzare le risorse collettive e che, se ben realizzato, massimizza l'utilità globale.

Esiste in effetti uno schema di meccanismo, noto come meccanismo di **Vickrey–Clarke–Groves**, o VCG, che gode di due proprietà favorevoli. Primo, massimizza l'utilità globale, data dalla somma delle utilità di tutte le parti, $\sum_i v_i$. Secondo, è rivelatore: per tutti gli agenti la strategia dominante è rivelare la propria valutazione. Non è necessario impegnarsi in complicati calcoli sulle strategie di offerta.

Facciamo un esempio considerando il problema dell'allocazione di un bene collettivo. Supponiamo che una città decida di installare dei dispositivi wireless per l'accesso gratuito a Internet. Il numero dei dispositivi disponibili è però minore del numero dei quartieri che ne desiderano uno. L'amministrazione cittadina vuole massimizzare l'utilità complessiva ma se domandasse a ogni consiglio di quartiere "quanto vale per voi un dispositivo wireless per l'accesso gratuito a Internet? (lo forniremo a chi lo apprezza di più)" ogni quartiere sarebbe incentivato a dichiarare un valore molto alto. Il meccanismo VCG scoraggia il ricorso a questo espediente e dà agli agenti un incentivo a dichiarare la propria vera valutazione. Funziona nel modo seguente,

1. Il centro chiede a ogni agente di dichiarare la propria valutazione di un dato bene, v_i .
2. Il centro assegna i beni a un insieme di vincitori, W , che massimizza $\sum_{i \in W} v_i$.
3. Il centro calcola per ogni agente vincitore quale perdita la sua presenza nel gioco ha determinato per i perdenti (ognuno dei quali ha ottenuto un'utilità nulla mentre vincendo avrebbe ottenuto v_j).
4. Ogni agente vincitore paga al centro un'imposta pari a tale perdita.

Supponiamo per esempio che siano disponibili 3 dispositivi e che gli interessati siano 5 e offrano 100, 50, 40, 20 e 10. L'insieme W dei 3 vincitori sarà composto dagli agenti che hanno offerto 100, 50 e 40 e l'utilità globale prodotta da questa assegnazione dei beni è 190. Se uno dei vincitori non avesse preso parte al gioco, sarebbe risultata vincente l'offerta 20, e lo stesso vale per gli altri vincitori. Quindi, ogni vincitore paga al centro un'imposta pari a 20.

Tutti i vincitori dovrebbero essere soddisfatti perché pagano un'imposta minore della loro valutazione e tutti i perdenti hanno ottenuto il massimo possibile per loro, dato che valutano i beni meno dell'imposta richiesta. Ecco perché il meccanismo è rivelatore. In questo esempio il valore cruciale è 20; sarebbe irrazionale per un agente offrire più di 20 se la sua valutazione è inferiore a 20, e viceversa. Poiché il valore cruciale non è noto (dipendendo dalle offerte altrui), è quindi sempre irrazionale fare un'offerta che non corrisponde alla propria valutazione.

Il meccanismo VCG è molto generale e può essere applicato a ogni tipo di gioco, non solamente alle aste, con una leggera generalizzazione del meccanismo appena descritto. Per

tragedia dei beni comuni

esternalità

Vickrey–Clarke–Groves

esempio, in un'**asta combinatoria** sono disponibili molti beni differenti e ogni partecipante può fare più offerte, per diversi sottoinsiemi di beni: in un'asta di lotti di terreno un offerente potrebbe essere interessato al lotto X o al lotto Y ma non a entrambi; un altro potrebbe desiderare tre lotti adiacenti; e così via. Il meccanismo VCG può essere utilizzato per individuare il risultato ottimo, sebbene con 2^N sottoinsiemi di N beni il calcolo del risultato ottimo sia NP-completo. Con alcuni accorgimenti, si ha che il meccanismo VCG è unico: qualsiasi altro meccanismo ottimo è sostanzialmente equivalente.

18.4.3 Votazione

teoria della scelta sociale

Esaminiamo ora un altro tipo di meccanismi: le procedure di voto, del tipo utilizzato per le decisioni politiche nelle società democratiche. Lo studio delle procedure di voto deriva dal campo della **teoria della scelta sociale**.

L'impostazione di base è la seguente. Come di consueto, abbiamo un insieme di agenti $N = \{1, \dots, n\}$, che in questo contesto saranno votanti. I votanti vogliono decidere rispetto a un insieme $\Omega = \{\omega_1, \omega_2, \dots\}$ di possibili risultati. In un'elezione politica, ogni elemento di Ω potrebbe rappresentare la vittoria di un diverso candidato.

Ogni votante avrà delle preferenze rispetto a Ω , che solitamente non sono espresse come utilità quantitative bensì come confronti qualitativi: scriviamo $\omega >_i \omega'$ per indicare che il risultato ω è valutato meglio del risultato ω' da parte dell'agente i . In un'elezione con tre candidati, l'agente i potrebbe avere $\omega_2 >_i \omega_3 >_i \omega_1$.

Il problema fondamentale della teoria della scelta sociale è combinare queste preferenze, utilizzando una **funzione di benessere sociale**, per individuare un **ordine di preferenza sociale**: una classificazione dei candidati, dal più apprezzato al meno apprezzato. In alcuni casi saremo interessati solamente a un **risultato sociale**: il risultato più apprezzato dal gruppo nel suo complesso. Scriveremo $\omega >^* \omega'$ per indicare che ω è classificato al di sopra di ω' nell'ordine di preferenza sociale.

Uno scenario più semplice è quello in cui non siamo interessati a ottenere un ordinamento completo dei candidati ma solamente a scegliere un insieme di vincitori. Una **funzione di scelta sociale** riceve come input un ordine di preferenza per ciascun votante e restituisce un insieme di vincitori.

Le società democratiche desiderano un risultato sociale che rifletta le preferenze dei votanti. Purtroppo ciò non è sempre semplice. Consideriamo il **paradosso di Condorcet**, un famoso esempio proposto dal Marchese di Condorcet (1743–1794). Supponiamo di avere tre risultati, $\Omega = \{\omega_a, \omega_b, \omega_c\}$ e tre votanti, $N = \{1, 2, 3\}$, aventi le seguenti preferenze:

$$\begin{aligned} \omega_a &>_1 \omega_b & >_1 \omega_c \\ \omega_c &>_2 \omega_a & >_2 \omega_b \\ \omega_b &>_3 \omega_c & >_3 \omega_a. \end{aligned} \tag{18.2}$$

Supponiamo ora di dover scegliere uno dei tre candidati sulla base di queste preferenze. Il paradosso è che:

- 2/3 dei votanti preferiscono ω_c a ω_a .
- 2/3 dei votanti preferiscono ω_a a ω_b .
- 2/3 dei votanti preferiscono ω_b a ω_c .

funzione di scelta sociale

paradosso di Condorcet



Per ogni possibile candidato, quindi, possiamo indicarne un altro che sarebbe preferito al primo per almeno 2/3 dell'elettorato. È ovvio che in una democrazia non si può sperare di accontentare *tutti* i votanti. Ciò dimostra che esistono scenari in cui *indipendentemente dal risultato scelto, una maggioranza dei votanti preferirà un risultato differente*. Una domanda naturale è se esista una “buona” procedura di scelta sociale che rifletta realmente le preferenze dei votanti. Per rispondere è necessario essere precisi su ciò che si intende quando si

dice che una regola è “buona”. Elenchiamo alcune proprietà che vorremmo caratterizzassero una buona funzione di benessere sociale:

- *Condizione di Pareto*: afferma semplicemente che se ogni votante valuta ω_i migliore di ω_j , allora $\omega_i >^* \omega_j$.
- *Condizione del vincitore di Condorcet*: si dice che un risultato è vincente secondo Condorcet se una maggioranza dei votanti lo preferisce a ogni altro risultato. In altri termini, un vincitore di Condorcet è un candidato che in un confronto a due batterebbe ogni altro candidato. Questa condizione afferma che, se ω_i è un vincitore di Condorcet, allora ω_i deve essere primo nella graduatoria.
- *Indipendenza delle alternative irrilevanti (IIA)*, dall'inglese *independence of irrelevant alternatives*): supponiamo che ci siano più candidati, tra cui ω_i e ω_j , e che le preferenze dei votanti siano tali che $\omega_i >^* \omega_j$. Supponiamo poi che un votante cambi le proprie preferenze, ma *non* rispetto alla valutazione relativa di ω_i e ω_j . La condizione IIA afferma che $\omega_i >^* \omega_j$ non deve cambiare.
- *Assenza di dittatura*: non deve accadere che la funzione di benessere sociale rifletta semplicemente le preferenze di uno dei votanti e ignori quelle di tutti gli altri.

Queste quattro condizioni sembrano ragionevoli, ma un teorema fondamentale della teoria della scelta sociale, chiamato teorema di Arrow (in riferimento a Kenneth Arrow), ci dice che è impossibile soddisfarle tutte e quattro (nei casi in cui i risultati sono almeno tre). Ciò significa che per ogni meccanismo di scelta sociale ci saranno situazioni (forse inconsuete o patologiche) che conducono a risultati controversi. Tuttavia, non significa che i processi di decisione democratici siano destinati a essere inadeguati. Non abbiamo ancora visto alcuna concreta procedura di voto; lo faremo ora.

- Con due soli candidati, il **voto a maggioranza semplice** (il modello standard negli USA e nel Regno Unito) è il meccanismo preferito. Si chiede a ogni votante quale dei due candidati preferisce e chi riceve più voti è il vincitore. voto a maggioranza semplice
- Con più di due risultati, un sistema comune è il **voto a maggioranza relativa (plurality voting)**. Si chiede a ogni votante quale sia la sua prima scelta e si seleziona il candidato (o i candidati, in caso di pareggio) che ottiene più voti, anche se nessuno ottiene la maggioranza assoluta dei voti. Sebbene sia comune, questo sistema di voto è stato criticato perché conduce a risultati impopolari. Un problema fondamentale è che esso prende in considerazione solamente il candidato meglio piazzato nelle preferenze di ciascun votante. voto a maggioranza relativa
- Il **conteggio di Borda** (da Jean-Charles de Borda, contemporaneo e rivale di Condorcet) è una procedura di voto che prende in considerazione tutte le informazioni contenute nell'ordinamento di preferenze di un votante. Supponiamo di avere k candidati. Per ogni votante i consideriamo l'ordine di preferenza $>_i$ e assegniamo il punteggio k al candidato meglio classificato, il punteggio $k - 1$ al secondo classificato e così via fino al candidato meno apprezzato nell'ordinamento di i . Il punteggio totale di ciascun candidato è il suo conteggio di Borda, e per ottenere il risultato sociale $>^*$, si ordinano i risultati in base ai rispettivi conteggi di Borda, dal più alto al più basso. Un problema pratico di questo sistema è che richiede ai votanti di esprimere preferenze su tutti i candidati, mentre alcuni votanti potrebbero essere interessati solo a un sottoinsieme di candidati. conteggio di Borda
- Nel **voto per approvazione**, i votanti indicano il sottoinsieme di candidati che approvano. Il vincitore (o i vincitori) è il candidato approvato dal maggior numero di votanti. Questo sistema viene spesso utilizzato quando si tratta di scegliere più vincitori. voto per approvazione
- Nel **voto alternativo** (in inglese, *instant runoff voting*), i votanti classificano tutti i candidati e, se un candidato è la prima scelta di più della metà dei votanti, viene dichiarato vincitore. In caso contrario, il candidato indicato come primo dal minor numero di votanti viene eliminato e rimosso da tutte le graduatorie di preferenza (in modo che i votanti che ave-

voto maggioritario

vano come prima scelta il candidato eliminato ora hanno come prima scelta un candidato diverso) e la procedura si ripete. Alla fine, un candidato avrà la maggioranza assoluta delle preferenze (a meno di un pareggio).

- Nel **voto maggioritario** (*true majority rule voting*), il vincitore è il candidato che batte ogni altro candidato nei confronti a due. Ai votanti viene chiesta una classificazione di tutti i candidati in ordine di preferenza. Diciamo che ω batte ω' se i votanti per i quali $\omega > \omega'$ sono di più di quelli per cui $\omega' > \omega$. Questo sistema ha la desiderabile proprietà che la maggioranza concorda sempre sul vincitore, ma anche la proprietà non desiderabile che in alcune elezioni non si raggiunge una decisione: nel paradosso di Condorcet, per esempio, nessun candidato ottiene la maggioranza.

teorema di Gibbard–Satterthwaite**Manipolazione strategica**

Oltre al teorema di Arrow, nell'area della teoria della scelta sociale esiste un altro importante risultato sfavorevole: il **teorema di Gibbard–Satterthwaite**. Questo risultato riguarda le circostanze nelle quali un votante può trarre beneficio dal *comunicare scorrettamente le proprie preferenze*.

Ricordiamo che una funzione di scelta sociale riceve come input un ordine di preferenza per ogni votante e fornisce come risultato un insieme di candidati vincenti. Ogni votante ha, ovviamente, le proprie vere preferenze ma nulla nella definizione di una funzione di scelta sociale richiede che i votanti dichiarino le loro preferenze in modo *veritiero*; possono dichiarare ciò che desiderano.

In alcuni casi, può essere sensato per un votante comunicare in modo non veritiero le proprie preferenze. Per esempio, nelle votazioni a maggioranza relativa, i votanti che pensano che il loro candidato preferito non abbia possibilità di vittoria potrebbero decidere di votare per la loro seconda scelta. Ciò significa che questo metodo è un gioco in cui i votanti devono pensare strategicamente (rispetto agli altri votanti) per massimizzare la propria utilità attesa.

Ciò solleva una domanda interessante: è possibile progettare un meccanismo di voto immune a una simile manipolazione? Un meccanismo rivelatore? Il teorema di Gibbard–Satterthwaite ci dice che non è possibile: *ogni funzione di scelta sociale che soddisfa la condizione di Pareto per un dominio con più di due risultati è manipolabile oppure dittoriale*. In altre parole, per ogni “ragionevole” procedura di scelta sociale esistono circostanze nelle quali un votante può in linea di principio trarre vantaggio da una dichiarazione non veritiera delle proprie preferenze. Tuttavia, il teorema non dice *come* una simile manipolazione potrebbe avvenire; e non dice che tale manipolazione sia probabile *nella pratica*.

**18.4.4 Contrattazione**

La contrattazione, o negoziazione, è un altro meccanismo usato frequentemente nella vita di ogni giorno. È oggetto di studio nell'ambito della teoria dei giochi fin dagli anni 1950 e più recentemente è diventata un'attività per agenti automatizzati. La contrattazione si utilizza quando gli agenti devono raggiungere un accordo su una questione di interesse comune. Gli agenti fanno delle offerte (o proposte di accordo) secondo specifici protocolli e accettano o rifiutano quelle altrui.

modello di contrattazione a offerte alternate**Contrattazione con il protocollo a offerte alternate**

Un importante protocollo di contrattazione è il **modello di contrattazione a offerte alternate**. Per semplicità assumeremo nuovamente che ci siano solo due agenti. La contrattazione si svolge con una sequenza di turni. Al turno 0, inizia A_1 facendo un'offerta. Se A_2 accetta l'offerta, essa viene implementata. Se A_2 rifiuta l'offerta la contrattazione passa al turno successivo. Questa volta A_2 fa un'offerta e A_1 sceglie se accettarla o declinare, e così via. Se la negoziazione non ha termine (perché gli agenti rifiutano ogni offerta) definiamo il risultato

esito conflittuale (*conflict deal*). Un'ipotesi utilmente semplificatrice è che entrambi gli agenti preferiscano raggiungere un risultato, uno qualsiasi, in un tempo finito piuttosto che rimanere bloccati per un tempo infinito nella situazione conflittuale.

Consideriamo come esempio la **suddivisione di una torta** per illustrare le offerte alternate. Il concetto è che vi sia una certa risorsa (la “torta”) il cui valore è 1, che può essere divisa in due parti, una per ciascun agente. Quindi in questo scenario un’offerta è data da una coppia di valori $(x, 1 - x)$, dove x è la porzione di torta ricevuta da A_1 e $1 - x$ è la porzione ricevuta da A_2 . Lo spazio dei possibili accordi (l'**insieme di negoziazione**) è quindi:

$$\{(x, 1 - x) : 0 \leq x \leq 1\}.$$

In questo scenario, come dovrebbero negoziare gli agenti? Per comprendere la risposta a questa domanda, consideriamo dapprima alcuni casi più semplici.

Supponiamo inizialmente di ammettere *un solo turno*. A_1 fa una proposta e A_2 può accettarla (nel qual caso l'accordo viene messo in atto) oppure rifiutarla (nel qual caso si realizza l'esito conflittuale). Si tratta di un **gioco dell'ultimatum**. In questo caso succede che A_1 , il **primo a muovere**, ha tutto il potere. Supponiamo che A_1 proponga di prendere tutta la torta, ovvero $(1, 0)$. Se A_2 rifiuta, allora viene implementato l'esito conflittuale; dato che per definizione A_2 preferirebbe ricevere 0 piuttosto che questo esito, per A_2 è più conveniente accettare. Ovviamente A_1 non può ottenere qualcosa di meglio dell'intera torta. Quindi queste due strategie (A_1 propone di prendere per sé l'intera torta e A_2 accetta), formano un equilibrio di Nash.

Consideriamo ora il caso in cui siano consentiti esattamente *due turni* di negoziazione. Ora i rapporti di potere sono diversi: A_2 può semplicemente rifiutare la prima offerta, trasformando il gioco in un gioco a un solo turno in cui A_2 è il primo a muovere e ottiene quindi tutta la torta. In generale, se il numero di turni è stabilito, allora chi muove per ultimo otterrà tutta la torta.

Passiamo ora al caso generale, in cui il numero di turni è *illimitato*. Supponiamo che A_1 adotti la strategia seguente:

Proporre sempre $(1, 0)$ e rifiutare sempre le controfferte.

Qual è per A_2 il modo migliore per rispondere? Se A_2 rifiuta continuamente la proposta, la contrattazione proseguirà all’infinito, che per definizione è il peggior risultato possibile per A_2 (e anche per A_1). Quindi A_2 non può fare di meglio che accettare la prima proposta di A_1 . Anche questo è un equilibrio di Nash. Che cosa succede se, invece, A_1 utilizza la strategia:

Proporre sempre $(0,8, 0,2)$ e rifiutare ogni offerta.

Con un ragionamento simile possiamo vedere che per questa offerta e per ogni possibile offerta $(x, 1 - x)$ dell'insieme di negoziazione, esiste una coppia di strategie di negoziazione in equilibrio di Nash tali che il risultato sarà l'accordo al primo turno.



Agenti impazienti

Questa analisi ci dice che se non viene posto un limite al numero di turni allora esisterà un numero infinito di equilibri di Nash. Aggiungiamo allora una assunzione:

Per ogni risultato x e per ogni coppia di tempi t_1 e t_2 , dove $t_1 < t_2$, per entrambi gli agenti il risultato x al tempo t_1 è preferibile al risultato x al tempo t_2 .

In altre parole, gli agenti sono **impazienti**. Un approccio standard all’impazienza è utilizzare un **fattore di sconto** γ_i (cfr. Paragrafo 17.1.1) per ogni agente ($0 \leq \gamma_i < 1$). Supponiamo che in un dato momento della contrattazione l'agente i riceva l'offerta di una fetta di torta di dimensione x . Il valore della fetta x al tempo t è $\gamma_i^t x$. Quindi all'inizio della contrattazione (tempo 0), il valore è $\gamma_i^0 x = x$, e in ogni punto temporale successivo il valore della medesima

esito conflittuale

insieme di negoziazione

gioco dell'ultimatum

offerta sarà minore. Un valore più alto di γ_i (più prossimo a 1) significa quindi maggiore pazienza; un valore più basso significa meno pazienza.

Per analizzare il caso generale, consideriamo dapprima la contrattazione su periodi di tempo stabiliti, come sopra. Nel caso a un solo turno, l'analisi è come quella vista sopra: si ha semplicemente un gioco dell' ultimatum. Con *due* turni la situazione cambia, perché il valore della torta si riduce in base al fattore γ_i . Supponiamo che A_2 rifiuti la proposta iniziale di A_1 . Allora A_2 otterrà l'intera torta con un ultimatum nel secondo turno. Ma il *valore* della torta si sarà ridotto: per A_2 ora varrà solamente γ_2 . L'agente A_1 può tenere conto di questo fatto offrendo $(1 - \gamma_2, \gamma_2)$, un'offerta che A_2 potrebbe accettare, non potendo a questo punto ottenere più di γ_2 (se volessimo evitare il pareggio, potremmo trasformare l'offerta in $(1 - (\gamma_2 + \varepsilon), \gamma_2 + \varepsilon)$ con un piccolo valore di ε).

Quindi, la strategia di A_1 di offrire $(1 - \gamma_2, \gamma_2)$ e quella di A_2 di accettare sono in equilibrio di Nash. Con questo protocollo, i giocatori pazienti (quelli con γ_2 elevato) sono in grado di ottenere fette di torta più grandi: in questo scenario, la pazienza è effettivamente una virtù.

Ora consideriamo il caso generale, in cui il numero di turni non è limitato. Come nel caso a un solo turno, A_1 può confezionare una proposta che A_2 dovrebbe accettare non potendo ottenere di più dati i fattori di sconto. Risulta che A_1 otterrà:

$$\frac{1 - \gamma_2}{1 - \gamma_1 \gamma_2}$$

e A_2 otterrà il resto.

Negoziazione in domini orientati ai compiti

dominio orientato ai compiti

In questo paragrafo consideriamo la negoziazione nei **domini orientati ai compiti** (*task-oriented*). In un contesto di questo tipo esiste una serie di compiti che devono essere eseguiti e ogni compito viene inizialmente assegnato a un insieme di agenti. Per gli agenti potrebbe essere vantaggioso negoziare per distribuire diversamente i vari compiti. Per esempio, supponiamo che alcuni dei compiti richiedano l'utilizzo del tornio e altri della fresatrice e che ogni agente che utilizza una di queste macchine debba sostenere un significativo costo di avviamento. Allora sarebbe sensato per un agente proporre a un altro: “Devo preparare in ogni caso la fresatrice; che ne dici se io faccio i tuoi lavori di fresatura e tu i miei di tornitura?”.

A differenza dello scenario della contrattazione, si inizia con una allocazione iniziale, quindi in mancanza di un accordo gli agenti eseguiranno i compiti T_i^0 loro assegnati originariamente.

Per semplicità, ipotizziamo nuovamente che gli agenti siano solo due. Sia T l'insieme di tutti i compiti e indichiamo con (T_1^0, T_2^0) l'assegnamento iniziale dei compiti ai due agenti, al tempo 0. Ogni compito di T deve essere assegnato a uno e un solo agente. Ipotizziamo di avere una funzione di costo c , che a ogni insieme di compiti T' associa un numero reale positivo $c(T')$ che indica il costo per qualsiasi agente dello svolgimento dei compiti T' (assumiamo che il costo dipenda solo dai compiti da svolgere, non dall'agente che li svolge). La funzione di costo è monotona (l'aggiunta di altri compiti non riduce mai il costo) e il costo dell'inattività è zero: $c(\{\}) = 0$. Per esempio, supponiamo che il costo di avviamento per la fresatrice sia 10 e che ciascuna attività di fresatura costi 1; allora il costo di un insieme di due lavori di fresatura è 12, e il costo per un insieme di cinque è 15.

Un'offerta della forma (T_1, T_2) significa che l'agente i si impegnerà a eseguire l'insieme di attività T_i , al costo $c(T_i)$. L'utilità per l'agente i è l'importo che guadagna accettando l'offerta, ovvero la differenza tra il costo dell'esecuzione di questo nuovo insieme di compiti e quello dell'insieme assegnato inizialmente:

$$U_i((T_1, T_2)) = c(T_i^0) - c(T_i).$$

Un'offerta (T_1, T_2) è **individualmente razionale** se $U_i((T_1, T_2)) \geq 0$ per entrambi gli agenti. Se un accordo non è individualmente razionale, allora almeno uno degli agenti sarebbe in una condizione migliore svolgendo i compiti assegnatigli inizialmente.

individualmente razionale

L'insieme di negoziazione per i domini orientati ai compiti (assumendo la razionalità degli agenti) è l'insieme delle offerte che sono sia individualmente razionali sia Pareto-ottime. Non ha senso fare un'offerta individualmente irrazionale che verrà rifiutata, né fare un'offerta quando ne esiste una migliore che incrementa l'utilità per un agente senza danneggiare gli altri.

Il protocollo di concessione monotona

Il protocollo di negoziazione da noi considerato per i domini orientati ai compiti è noto come **protocollo di concessione monotona**. Le sue regole sono le seguenti.

protocollo di concessione monotona

- La negoziazione si svolge in una serie di turni.
- Al primo turno, i due agenti propongono *simultaneamente* degli accordi $D_i = (T_1, T_2)$, appartenenti all'insieme di negoziazione (mentre in precedenza abbiamo considerato un'alternanza di offerte).
- Viene raggiunto un accordo se i due agenti propongono rispettivamente D_1 e D_2 tali che (i) $U_1(D_2) \geq U_1(D_1)$ oppure (ii) $U_2(D_1) \geq U_2(D_2)$, ovvero se uno degli agenti trova che la proposta dell'altro è almeno altrettanto buona della propria. Se c'è accordo, allora la regola per determinare quale proposta verrà realizzata è la seguente: se ciascuna proposta è pari o superiore a quella dell'altro, allora se ne sceglie una a caso. Se solo una delle proposte equivale o supera quella dell'altro, allora è la proposta selezionata.
- Se non si raggiunge un accordo, allora la negoziazione prosegue con un altro turno di proposte simultanee. Nel turno $t + 1$, ogni agente deve ripetere la proposta del turno precedente oppure fare una **concessione**: una proposta più conveniente per l'altro agente (ovvero, di utilità maggiore).
- Se nessuno dei due fa concessioni, allora la negoziazione termina e gli agenti implementano l'esito conflittuale, svolgendo i compiti loro assegnati all'inizio.

concessione

Poiché l'insieme dei possibili accordi è finito, gli agenti non possono negoziare all'infinito: raggiungeranno un accordo oppure arriverà un turno in cui nessuno dei due farà concessioni. Il protocollo tuttavia non garantisce che l'accordo venga raggiunto *rapidamente*: poiché il numero dei possibili accordi è $O(2^{|T|})$, si comprende che la negoziazione continuerà per un numero di turni esponenziale nel numero di compiti da assegnare.

La strategia di Zeuthen

Finora non abbiamo detto nulla su come i negoziatori potrebbero o dovrebbero comportarsi quando si utilizza il protocollo di concessione monotona per domini orientati ai compiti. Una delle possibili strategie è la **strategia di Zeuthen**.

strategia di Zeuthen

L'idea alla base di questa strategia è misurare la *propensione di un agente a rischiare il conflitto*. Intuitivamente, un agente sarà più disposto a rischiare il conflitto se la differenza di utilità tra la sua proposta attuale e l'esito conflittuale è modesta. In tal caso l'agente ha poco da perdere se la negoziazione fallisce e si realizza l'esito conflittuale, quindi è più disposto a rischiare il conflitto e meno disposto a fare concessioni. All'opposto, se la differenza tra la proposta attuale dell'agente e l'esito conflittuale è grande, allora l'agente ha più da perdere, è meno propenso a rischiare il conflitto e più disposto a fare concessioni.

La propensione dell'agente i a rischiare il conflitto al turno t , indicata da $riscio'_i$, si misura come segue:

$$riscio'_i = \frac{\text{utilità che } i \text{ perde non facendo concessioni e accettando l'offerta di } j}{\text{utilità che } i \text{ perde non facendo concessioni e causando conflitto}}.$$

Finché non si raggiunge un accordo, il valore di $rischio_i^t$ sarà compreso tra 0 e 1. Valori più alti di $rischio_i^t$ (prossimi a 1) indicano che i ha meno da perdere ed è quindi più disposto a rischiare il conflitto.

Secondo la strategia di Zeuthen, la prima proposta di ogni agente dovrebbe appartenere all’insieme di negoziazione e massimizzare l’utilità dell’agente stesso (potrebbe esisterne più di una). Poi, al turno t della negoziazione, l’agente che dovrebbe fare una concessione è quello con la minore propensione al rischio, ovvero quello che rischia di perdere di più in caso di conflitto.

La successiva domanda cui rispondere è: quanto concedere? La risposta fornita dalla strategia di Zeuthen è “il minimo necessario per spostare l’equilibrio del rischio verso l’altro agente”. In altre parole, un agente dovrebbe fare la *minore* concessione in grado di fare in modo che l’altro agente faccia una concessione al turno successivo.

C’è un ultimo ritocco da fare alla strategia di Zeuthen: supponiamo che a un certo punto i due agenti abbiano rischi *uguali*. Allora, secondo la strategia, entrambi dovrebbero concedere. Ma sapendo ciò, uno degli agenti potrebbe “deviare” e non fare alcuna concessione, ottenendo un beneficio. Per evitare la possibilità che a questo punto entrambi facciano concessioni, integriamo la strategia stabilendo che gli agenti “tirino a sorte” per decidere chi debba concedere ogni volta che si trovano in situazione di rischio equivalente.

Con questa strategia l’accordo sarà Pareto-ottimo e individualmente razionale. Tuttavia, dato che lo spazio dei possibili accordi è esponenziale nel numero di compiti da svolgere, seguire questa strategia può comportare $O(2^{|T|})$ calcoli della funzione di costo a ogni passaggio della negoziazione. Infine, la strategia di Zeuthen (con la regola del lancio di moneta) è in equilibrio di Nash.

18.5 Riepilogo

- La **pianificazione multiagente** è necessaria quando nell’ambiente sono presenti altri agenti con cui cooperare o competere. Si possono costruire piani congiunti, che devono però essere accompagnati da qualche forma di coordinamento se due agenti devono concordare sulla scelta del piano congiunto da eseguire.
- La **teoria dei giochi** descrive il comportamento razionale nelle situazioni in cui interagiscono più agenti. È per le decisioni multiagente ciò che la teoria delle decisioni è per le decisioni di un singolo agente.
- I **concetti di soluzione** nella teoria dei giochi caratterizzano i risultati razionali di un gioco, ovvero i risultati che si manifestano se tutti gli agenti agiscono razionalmente.
- La **teoria dei giochi non cooperativi** assume che gli agenti debbano prendere le rispettive decisioni in modo indipendente. L'**equilibrio di Nash** è il concetto di soluzione più importante nella teoria dei giochi non cooperativi. Un equilibrio di Nash è un profilo di strategie tale che nessun agente è incentivato a cambiare la propria strategia. Esistono tecniche dedicate ai giochi ripetuti e a quelli sequenziali.
- La **teoria dei giochi cooperativi** considera scenari in cui gli agenti possono fare accordi vincolanti per formare coalizioni allo scopo di cooperare. Nei giochi cooperativi, i concetti di soluzione tentano di individuare le coalizioni stabili (il **nucleo**) e il modo per suddividere equamente il valore che una coalizione ottiene (il **valore di Shapley**).
- Per alcune importanti classi di decisioni multiagente esistono tecniche specifiche: il contract net protocol per l’assegnazione di compiti; le aste per allocare in modo efficiente le risorse scarse; la contrattazione per raggiungere accordi su questioni di interesse comune; e le procedure di voto per aggregare le preferenze.

Note storiche e bibliografiche

Una curiosità a proposito dell'IA è che i ricercatori non iniziarono a considerare seriamente le questioni riguardanti l'interazione tra agenti fino agli anni 1980; e il campo dei sistemi multiagente non si affermò realmente come sottodisciplina distinta dell'IA fino a un decennio più tardi. Tuttavia, idee che accennavano ai sistemi multiagente erano già presenti negli anni 1970. Per esempio, con la sua influente teoria della *società della mente*, Marvin Minsky (1986, 2007) propose che le menti umane siano costituite da un insieme di agenti. Doug Lenat aveva idee simili come parte di un sistema che denominò BEINGS (Lenat, 1975). Negli anni 1970, sviluppando il proprio dottorato sul sistema PLANNER, Carl Hewitt propose un modello di computazione come interazione di agenti chiamato **actor model**, affermatosi poi come uno dei modelli fondamentali nel calcolo concorrente (Hewitt, 1977; Agha, 1986).

La preistoria del campo dei sistemi multiagente è documentata esaurientemente in una serie di studi intitolata *Readings in Distributed Artificial Intelligence* (Bond e Gasser, 1988). La raccolta è preceduta da una enunciazione dettagliata delle principali sfide della ricerca sui sistemi multiagente, rimasta notevolmente attuale dopo oltre trent'anni. Le prime ricerche sui sistemi multiagente tendevano ad assumere che tutti gli agenti in un sistema agissero secondo un interesse comune, con un unico progettista. Questo è oggi riconosciuto come caso speciale del più generale scenario multiagente; il caso speciale è noto come **problem solving cooperativo distribuito**. Un sistema fondamentale di quest'epoca era il Distributed Vehicle Monitoring Testbed o DVMT, sviluppato sotto la supervisione di Victor Lesser all'Università del Massachusetts (Lesser e Corkill, 1988). Il DVMT modellava uno scenario in cui una serie di agenti, dei sensori acustici distribuiti geograficamente, cooperano per rilevare il movimento dei veicoli.

L'era contemporanea della ricerca sui sistemi multiagente iniziò alla fine degli anni 1980, quando ci si rese conto che la norma è che gli agenti abbiano preferenze differenti, nell'IA come nella società. Da quel momento, la teoria dei giochi iniziò ad affermarsi come metodologia principale per lo studio di tali agenti.

La pianificazione multiagente ha raggiunto la popolarità in anni recenti, sebbene abbia una storia lunga. Konolige (1982) formalizzò la pianificazione multiagente nella logica del primo ordine, mentre Pednault

(1986) ne diede una descrizione di tipo STRIPS. La nozione di intenzione congiunta, essenziale se gli agenti devono eseguire un piano congiunto, deriva dal lavoro sugli atti comunicativi (Cohen e Perrault, 1979; Cohen e Levesque, 1990; Cohen *et al.*, 1990). Boutilier e Brafman (2001) mostrarono come adattare la pianificazione con ordinamento parziale a uno scenario multiattore. Brafman e Domshlak (2008) individuarono un algoritmo di pianificazione multiattore la cui complessità cresce solo linearmente con il numero di attori, a condizione che il grado di accoppiamento (misurato parzialmente dalla larghezza d'albero del grado delle interazioni tra gli agenti) sia limitato.

La pianificazione multiagente è più difficile quando sono presenti agenti antagonisti. Come disse Jean-Paul Sartre (1960): "In una partita di calcio, tutto è reso più complicato dalla presenza dell'altra squadra". Il generale Dwight D. Eisenhower diceva: "Quando ci si prepara a una battaglia i piani sono inutili, ma pianificare è indispensabile", nel senso che è importante avere un piano o una politica condizionali, e non aspettarsi che un piano non condizionale abbia successo.

Il tema dell'apprendimento per rinforzo distribuito e multiagente (RL, da *reinforcement learning*) non è stato trattato in questo capitolo ma attualmente è di grande interesse. Nel RL distribuito, lo scopo è individuare metodi mediante i quali più agenti coordinati imparano a ottimizzare una funzione di utilità comune. Per esempio, è possibile individuare metodi che permettano a dei sottoagenti distinti, dedicati alla navigazione robotica e al rilevamento degli ostacoli, di realizzare collaborativamente un sistema di controllo globalmente ottimo? Alcuni semplici risultati in questa direzione sono stati ottenuti (Guestrin *et al.*, 2002; Russell e Zimdars, 2003). L'idea base è che ogni sottoagente apprenda la propria funzione Q (una sorta di funzione di utilità; cfr. il Paragrafo 22.3.3 del Volume 2) dal flusso di ricompense che riceve. Per esempio, un componente di navigazione robotica può ricevere ricompense quando compie degli avanzamenti verso l'obiettivo, mentre il componente di rilevamento degli ostacoli riceve ricompense negative a ogni collisione. Ogni decisione globale massimizza la somma delle funzioni Q e il processo complessivamente converge verso soluzioni globalmente ottime.

Le radici della teoria dei giochi risalgono al XVII secolo, con le proposte di Christiaan Huygens e Gottfried

Leibniz di studiare scientificamente e matematicamente le interazioni umane competitive e cooperative. Durante tutto il XIX secolo, molti economisti di primo piano hanno creato semplici modelli matematici per analizzare particolari esempi di situazioni competitive.

I primi risultati formali in teoria dei giochi si devono a Zermelo (1913) (il quale l'anno precedente aveva suggerito una forma di ricerca minimax per i giochi, seppure non corretta). Emile Borel (1921) introdusse la nozione di strategia mista. John von Neumann (1928) dimostrò che ogni gioco a due persone e a somma zero ha un equilibrio maximin con strategie miste e un valore ben definito. La collaborazione di Von Neumann con l'economista Oskar Morgenstern portò alla pubblicazione nel 1944 di *Theory of Games and Economic Behavior* (“Teoria dei giochi e del comportamento economico”), il libro che definì la teoria dei giochi. La pubblicazione del libro fu ritardata dalla scarsità della carta durante la guerra, finché un membro della famiglia Rockfeller non la finanziò personalmente.

Nel 1950, all'età di 21 anni, John Nash pubblicò le sue idee sugli equilibri nei giochi generali (non a somma zero). La sua definizione di soluzione di equilibrio, sebbene anticipata nell'opera di Cournot (1838), divenne nota come equilibrio di Nash. Dopo un lungo rinvio, dovuto alla schizofrenia di cui soffrì a partire dal 1959, Nash ricevette il premio Nobel per l'economia (assieme a Reinhart Selten e John Harsanyi) nel 1994. L'equilibrio di Bayes-Nash fu descritto da Harsanyi (1967) e discusso da Kadane e Larkey (1982). Alcune questioni riguardanti l'impiego della teoria dei giochi per il controllo degli agenti sono trattate da Binmore (1982). Aumann e Brandenburger (1995) mostraronno come si possano raggiungere equilibri differenti a seconda delle conoscenze di cui ciascun giocatore dispone.

Il dilemma del prigioniero venne inventato come esercizio didattico da Albert W. Tucker nel 1950 (sulla base di un esempio di Merrill Flood e Melvin Dresher) ed è stato trattato ampiamente da Axelrod (1985) e Poundstone (1993). I giochi ripetuti vennero introdotti da Luce e Raiffa (1957), e Abreu e Rubinstein (1988) discutono di come utilizzare in questi giochi le macchine a stati finiti (teoricamente, **macchine di Moore**). Il testo di Mailath e Samuelson (2006) si concentra sui giochi ripetuti.

I giochi a informazione parziale in forma estesa vennero introdotti da Kuhn (1953). La forma sequenziale per i giochi a informazione parziale fu inventata da Romanovskii (1962) e, in modo indipendente, da Koller *et al.* (1996); lo studio di Koller e Pfeffer (1997)

è un'introduzione accessibile a questo campo e descrive un sistema per rappresentare e risolvere i giochi sequenziali.

L'utilizzo dell'astrazione per ridurre un albero di gioco a una dimensione che consenta la soluzione con la tecnica di Koller è stato introdotto da Billings *et al.* (2003). In seguito, metodi migliori per trovare gli equilibri hanno consentito di risolvere astrazioni con 10^{12} stati (Gilpin *et al.*, 2008; Zinkevich *et al.*, 2008). Bowling *et al.* (2008) spiegano come utilizzare il campionamento di importanza per ottenere una migliore stima del valore di una strategia. Waugh *et al.* (2009) hanno scoperto che l'approccio dell'astrazione è suscettibile di errori sistematici nell'approssimazione della soluzione di equilibrio: funziona per alcuni giochi ma non per altri. Brown e Sandholm (2019) hanno dimostrato, almeno nel caso del poker Texas hold 'em a più giocatori, che questi punti deboli possono essere superati se si ha sufficiente capacità di calcolo. Impegnando un server a 64 core per 8 giorni hanno calcolato una strategia di base per il programma Pluribus, con la quale sono riusciti a battere dei campioni umani.

La teoria dei giochi di Markov, chiamati anche giochi stocastici, combina la teoria dei giochi e gli MDP (Littman, 1994; Hu e Wellman, 1998). Shapley (1953b) descrisse l'algoritmo di iterazione dei valori indipendentemente da Bellman, ma i suoi risultati non furono ampiamente apprezzati, forse perché vennero presentati nel contesto dei giochi di Markov. La teoria evolutiva dei giochi (Smith, 1982; Weibull, 1995) studia il mutamento delle strategie nel tempo: se la strategia del rivale cambia, come reagire?

Libri di testo che guardano alla teoria dei giochi da un punto di vista economico sono quelli di Myerson (1991), Fudenberg e Tirole (1991), Osborne (2004) e Osborne e Rubinstein (1994). Per una prospettiva legata all'IA abbiamo Nisan *et al.* (2007) e Leyton-Brown e Shoham (2008). Un'utile panoramica sulle decisioni multiagente si trova in Sandholm (1999).

Il RL multiagente si differenzia da quello distribuito per la presenza di agenti che non possono coordinare le loro azioni (se non con atti comunicativi esplicativi) e che potrebbero non avere la stessa funzione di utilità.

Quindi, il RL multiagente si occupa di problemi di teoria dei giochi sequenziali o **giochi di Markov**, definiti nel Capitolo 17. A causare problemi è il fatto che, mentre un agente sta apprendendo come battere la politica del proprio avversario, l'avversario modifica la propria politica per battere l'agente. Perciò, l'equilibrio è **non stazionario** (cfr. Paragrafo 13.4.2).

Littman (1994) notò questa difficoltà introducendo i primi algoritmi di RL per i giochi di Markov a somma zero. Hu e Wellman (2003) presentarono un algoritmo Q-learning per giochi generali che converge quando l'equilibrio di Nash è unico; quando invece sono presenti più equilibri, la nozione di convergenza non è così semplice da definire (Shoham *et al.*, 2004).

I giochi di assistenza sono stati introdotti, con il termine **apprendimento per rinforzo inverso cooperativo**, da Hadfield-Menell *et al.* (2017a). Malik *et al.* (2018) ha introdotto un efficiente risolutore di POMDP pensato specificamente per i giochi di assistenza. Questi giochi sono legati ai **giochi principal-agent** studiati in economia, in cui un capo (per esempio un datore di lavoro) e un agente (per esempio un dipendente) devono trovare un accordo reciprocamente vantaggioso nonostante abbiano preferenze fortemente differenti. Le differenze principali sono che (1) il robot non ha preferenze proprie e (2) il robot non ha certezza riguardo alle preferenze umane che deve ottimizzare.

I giochi cooperativi vennero studiati inizialmente da von Neumann e Morgenstern (1944). La nozione di nucleo venne introdotta da Donald Gillies (1959) e il valore di Shapley da Lloyd Shapley (1953a). Una buona introduzione alla matematica dei giochi cooperativi è quella di Peleg and Sudholter (2002). I giochi semplici sono discussi nel dettaglio da Taylor e Zwicker (1999). Un'introduzione agli aspetti computazionali della teoria dei giochi cooperativi si trova in Chalakiadakis *et al.* (2011).

Negli ultimi tre decenni sono stati sviluppati molti schemi di rappresentazione compatti per i giochi cooperativi, a partire dal lavoro di Deng e Papadimitriou (1994). Il più influente tra questi schemi è il modello delle reti di contributi marginali, introdotto da Jeong e Shoham (2005). L'approccio alla formazione delle coalizioni che abbiamo descritto è stato sviluppato da Sandholm *et al.* (1999); Rahwan *et al.* (2015) esaminano lo stato dell'arte.

Il contract net protocol venne introdotto da Reid Smith con la sua tesi di dottorato all'Università di Stanford alla fine degli anni 1970 (Smith, 1980). Il protocollo appare così naturale che viene regolarmente reinventato ancora oggi. Le basi economiche del protocollo sono state studiate da Sandholm (1993).

Le aste e la progettazione di meccanismi sono stati argomenti di primo piano nella scienza informatica e nella IA per diversi decenni: in Nisan (2007) vengono trattati da una prospettiva informatica convenzionale, in Krishna (2002) si trova un'introduzione alla teo-

ria delle aste e Cramton *et al.* (2006) presentano una raccolta di articoli sugli aspetti computazionali delle aste.

Il premio Nobel per l'economia 2007 venne assegnato a Hurwicz, Maskin e Myerson "per avere posto le fondamenta della teoria della progettazione di meccanismi" (Hurwicz, 1973). La tragedia dei beni comuni, uno dei problemi ispiratori di questo campo, venne analizzato da William Lloyd (1833) ma fu denominato e portato all'attenzione pubblica da Garrett Hardin (1968). Ronald Coase presentò un teorema secondo cui, se le risorse sono di proprietà privata e se i costi di transazione sono abbastanza bassi, allora le risorse verranno gestite in modo efficiente (Coase, 1960). Coase sottolinea che, nella pratica, i costi di transazione sono alti, quindi questo teorema non trova applicazione e si dovrebbero cercare altre soluzioni al di là della privatizzazione e del mercato. *Governing the Commons* di Elinor Ostrom's (1990) descrive soluzioni a questo problema basate sull'assegnazione del potere di controllo sulla gestione delle risorse alle persone del luogo, che hanno le maggiori conoscenze sulla situazione. Sia Coase che Ostrom vinsero il premio Nobel per l'economia per il loro lavoro.

Il principio di rivelazione si deve a Myerson (1986) e il teorema dell'equivalenza dei ricavi è stato sviluppato in modo indipendente da Myerson (1981) e da Riley e Samuelson (1981). Due economisti, Milgrom (1997) e Klemperer (2002), scrivono delle aste multi-miliardarie per l'assegnazione delle frequenze in cui erano coinvolti.

La progettazione di meccanismi è utilizzata nella pianificazione (Hunsberger e Grosz, 2000; Stone *et al.*, 2009) e nello scheduling (Rassenti *et al.*, 1982) multagiante. Varian (1995) fornì una succinta panoramica con collegamenti alla letteratura informatica e Rosenschein e Zlotkin (1994) presentarono una trattazione con applicazioni nella IA distribuita. Altri lavori connessi all'IA distribuita assumono varie denominazioni, tra cui intelligenza collettiva (Tumer e Wolpert, 2000; Segaran, 2007) e controllo mediante il mercato (Clearwater, 1996). Dal 2001 si tiene la competizione annuale Trading Agents Competition (TAC), in cui degli agenti tentano di realizzare il profitto più alto in una serie di aste (Wellman *et al.*, 2001; Arunachalam e Sadeh, 2005).

La letteratura sulla scelta sociale è enorme e spazia da considerazioni filosofiche sulla natura della democrazia ad analisi altamente tecniche di specifiche procedure di voto. Campbell e Kelly (2002) è un buon punto di partenza rispetto a questa letteratura. Il testo

Handbook of Computational Social Choice contiene una serie di articoli che esplorano i temi e i metodi della ricerca in questo campo (Brandt *et al.*, 2016). Il teorema di Arrow elenca le proprietà desiderabili dei sistemi di voto e dimostra che è impossibile ottenerle tutte (Arrow, 1951). Dasgupta e Maskin (2008) hanno mostrato che il sistema di voto a maggioranza (non a maggioranza relativa né con classificazione delle scelte) è il più solido. La complessità di calcolo della manipolazione delle elezioni è stata studiata per la prima volta da Bartholdi *et al.* (1989).

Abbiamo considerato solo superficialmente il lavoro sulla negoziazione nella pianificazione multiagente. Durfee e Lesser (1989) discussero di come più agenti possono dividere i compiti mediante la negoziazione. Kraus *et al.* (1991) descrisse un sistema per giocare a Diplomacy, un gioco da tavolo che richiede negoziazione, formazione di coalizioni e disonestà. Stone (2000) mostrò come degli agenti possano cooperare come compagni di squadra nell'ambiente competitivo, dinamico e parzialmente osservabile del calcio robotico. In un articolo successivo, Stone (2003) analizzò due ambienti multiagente competitivi (RoboCup,

una competizione di calcio robotico, e TAC, la gara di aste Trading Agents Competition) scoprendo che l'intrattabilità computazionale degli attuali approcci teoricamente ben fondati hanno fatto sì che molti sistemi multiagente siano progettati con metodi *ad hoc*. Sarit Kraus ha sviluppato una serie di agenti in grado di negoziare con le persone e con altri agenti; si veda Kraus (2001) per una panoramica. Il protocollo di concessione monotona per la negoziazione automatizzata è stato proposto da Jeffrey S. Rosenschein e dai suoi studenti (Rosenschein e Zlotkin, 1994). Il protocollo a offerte alternate è stato sviluppato da Rubinstein (1982).

Tra i testi sui sistemi multiagente abbiamo Weiss (2000a), Young (2004), Vlassis (2008), Shoham e Leyton-Brown (2009) e Wooldridge (2009). La più importante conferenza sui sistemi multiagente è l'International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS); esiste anche una rivista con lo stesso nome. Anche l'ACM Conference on Electronic Commerce (EC) pubblica molti studi significativi, in particolare nell'area degli algoritmi per le aste. La principale rivista di teoria dei giochi è *Games and Economic Behavior*.

- 27.1 I limiti dell'IA
- 27.2 Le macchine possono davvero pensare?
- 27.3 L'etica dell'IA
- 27.4 Riepilogo

Filosofia, etica e sicurezza dell'IA

In cui esaminiamo le grandi domande relative al significato dell'IA, a come possiamo svilupparla e applicarla in modo etico, a come possiamo mantenerla sicura.

Da sempre i filosofi pongono domande di grande portata: come funziona la mente? È possibile che le macchine agiscano con intelligenza allo stesso modo delle persone? Tali macchine avrebbero menti reali e coscienti?

Aggiungiamo altre domande a nostra volta: quali sono le implicazioni etiche di un uso quotidiano delle macchine intelligenti? Alle macchine dovrebbe essere consentito di prendere la decisione di uccidere esseri umani? Gli algoritmi possono essere equi e non distorti? Che cosa faranno gli esseri umani, se le macchine saranno in grado di svolgere tutti i tipi di lavori? E come possiamo controllare macchine che potrebbero diventare più intelligenti di noi?

27.1 I limiti dell'IA

Nel 1980 il filosofo John Searle introdusse una distinzione tra l'**IA debole** – l'idea che le macchine potrebbero agire *come se* fossero intelligenti – e l'**IA forte** – l'asserzione che le macchine che fanno ciò siano *realmente* pensanti in modo cosciente (e non si limitino a *simulare* di pensare). Con il passare del tempo, la definizione di IA forte si è spostata verso la cosiddetta “IA di livello umano” o “IA generale”, costituita da programmi che sono in grado di risolvere qualsiasi varietà di compiti, anche nuovi, e di farlo bene come gli esseri umani.

I critici dell'IA debole, che non credevano alla possibilità che le macchine potessero avere un comportamento intelligente, oggi appaiono mopi come Simon Newcomb, che nell'ottobre 1903 scrisse: “Il volo aereo è uno dei tanti problemi che l'uomo non potrà mai risolvere”, soltanto due mesi prima che i fratelli Wright compirono il primo volo a Kitty Hawk. Tuttavia, i rapidi progressi compiuti in anni recenti non dimostrano che le possibilità dell'IA siano illimitate. Alan Turing (1950), il primo a definire l'IA, fu anche il primo a sollevare delle obiezioni, anticipando quasi tutte quelle poi sollevate da altri.

GOFAI

27.1.1 L'argomento dell'informalità del comportamento

L'argomento di Turing “dell'informalità del comportamento” afferma che il comportamento umano è decisamente troppo complesso per poter essere catturato da qualsiasi insieme di regole formali – gli esseri umani utilizzano alcuni criteri guida informali che (secondo tale argomento) non potrebbero mai essere catturati in un insieme formale di regole e, quindi, non potrebbero mai essere codificati in un programma informatico.

Un importante fautore di questo punto di vista fu Hubert Dreyfus, che scrisse influenti opere di critica dell'intelligenza artificiale: *What Computers Can't Do* (1972), la sua continuazione *What Computers Still Can't Do* (1992) e, con suo fratello Stuart, *Mind Over Machine* (1986). In modo simile, il filosofo Kenneth Sayre (1993) disse: “L'intelligenza artificiale perseguita nel culto del *computationalismo* non ha la minima possibilità di produrre risultati durevoli”. La tecnologia oggetto di tali critiche venne chiamata **GOFAI** (*good old-fashioned AI*, traducibile in “IA alla vecchia maniera”).

La GOFAI corrisponde al più semplice agente logico descritto nel Capitolo 7, e abbiamo visto come sia difficile catturare ogni contingenza di un comportamento appropriato in un insieme di regole logiche necessarie e sufficienti – il cosiddetto **problema della qualificazione**. Tuttavia, come abbiamo visto nel Capitolo 12 del Volume 1, i sistemi di ragionamento probabilistico sono più appropriati per domini aperti e difficilmente circoscrivibili, e come viene mostrato nel Capitolo 21 del Volume 2, i sistemi di deep learning si comportano bene in una varietà di compiti “informali”. La critica, quindi, non è rivolta contro i computer *in sé*, ma contro un particolare stile di programmazione mediante regole logiche; tale stile era diffuso negli anni 1980, ma è stato poi eclissato da approcci nuovi.

Uno degli argomenti più forti di Dreyfus sostiene che gli agenti debbano essere entità situate nell'ambiente e non motori di inferenza logica incorporei. Un agente la cui comprensione di “cane” derivi soltanto da un insieme limitato di formule logiche come “*Cane(x)* \Rightarrow *Mammifero(x)*” è svantaggiato rispetto a un agente che ha osservato i cani correre, ha giocato con loro e si è fatto leccare. Come afferma il filosofo Andy Clark (1998), “I cervelli biologici sono prima di tutto sistemi di controllo per corpi biologici. E i corpi biologici si muovono e agiscono in ambienti reali”. Secondo Clark, siamo “bravi a giocare a frisbee, meno in logica”.

cognizione incarnata

L'approccio della **cognizione incarnata** (o incorporata) afferma che non ha senso considerare il cervello separatamente: la cognizione ha luogo all'interno di un corpo, il quale è inserito in un ambiente. Dobbiamo studiare il sistema nella sua interezza; il funzionamento del cervello sfrutta le regolarità nel suo ambiente, che comprende le altre parti del corpo. In questo approccio, la robotica, i sistemi di visione e altri sensori diventano centrali e non periferici.

In generale, Dreyfus ha individuato delle aree in cui l'IA non era in grado di fornire risposte complete e in base a questo ha affermato che l'IA fosse impossibile; oggi vediamo che in molte di quelle aree si svolgono continue ricerche e sviluppi che portano a un aumento delle capacità, non all'impossibilità.

27.1.2 L'argomento della disabilità

L’“argomento della disabilità” sostiene che “una macchina non potrà mai fare *X*”, e come esempi di *X*, Turing elenca le seguenti attività:

Essere gentile, pieno di risorse, bella, amichevole, avere iniziativa, senso dell'umorismo, riconoscere ciò che è giusto e sbagliato, commettere errori, innamorarsi, gustare fragole e gelato, far sì che qualcuno si innamori di essa, imparare dall'esperienza, usare le parole correttamente, essere l'oggetto del proprio pensiero, avere una diversità di comportamenti pari a quella di un essere umano, fare qualcosa di veramente nuovo.

In retrospettiva, alcune di queste attività sono piuttosto semplici, tutti abbiamo incontrato computer che “commettono errori”. I computer dotati di capacità di metaragionamento (Ca-

pitolo 5 del Volume 1) sono in grado di esaminare le elaborazioni che loro stessi hanno svolto, divenendo quindi l'oggetto del proprio pensiero. Esiste una tecnologia vecchia ormai di secoli che ha dimostrato di poter “far innamorare qualcuno”: l'orsacchiotto di pezza. L'esperto di scacchi David Levy prevede che nel 2050 le persone si innamoreranno spesso di robot umanoidi. E per quanto riguarda la possibilità che i robot si innamorino, si tratta di un tema comune nelle opere di fantasia e fantascienza,¹ che però non ha riscosso particolare attenzione in campo accademico (Kim *et al.*, 2007). I computer hanno fatto cose “veramente nuove” contribuendo a significative scoperte in astronomia, matematica, chimica, mineralogia, biologia, informatica e altri campi, e creando nuove forme d’arte attraverso il trasferimento di stili (Gatys *et al.*, 2016). In generale, i programmi superano le prestazioni umane in alcune attività e restano indietro in altre. L'unica cosa che chiaramente non possono fare è essere esattamente umani.

27.1.3 L'obiezione matematica

Turing (1936) e Gödel (1931) hanno dimostrato che esistono alcune questioni matematiche a cui particolari sistemi formali non possono dare risposta. Il teorema di incompletezza di Gödel (cfr. Paragrafo 9.5 del Volume 1) ne è l'esempio più famoso. Brevemente, in ogni sistema formale di assiomi F abbastanza potente da gestire l'aritmetica è possibile costruire una cosiddetta “formula di Gödel” $G(F)$ con le seguenti proprietà:

- $G(F)$ è una formula di F , ma non può essere dimostrata all'interno di F .
- Se F è consistente, allora $G(F)$ è vera.

Filosofi come J. R. Lucas (1961) hanno sostenuto che questo teorema dimostra che le macchine sono mentalmente inferiori agli esseri umani, perché in quanto sistemi formali sono limitate dal teorema di incompletezza: non possono stabilire il valore di verità della propria formula di Gödel, mentre gli esseri umani non hanno questa limitazione. Questa argomentazione ha causato enormi controversie, dando origine a una vasta letteratura che include due libri del matematico Sir Roger Penrose (1989, 1994). Penrose riprende l'argomento di Lucas aggiungendo nuovi particolari, come l'ipotesi che gli umani siano diversi perché i loro cervelli funzionano grazie alla gravità quantistica, una teoria che esprime false previsioni sulla fisiologia del cervello.

Esamineremo tre aspetti problematici dell'argomento di Lucas. In primo luogo un agente non dovrebbe vergognarsi troppo di non riuscire a stabilire la verità di una formula, anche quando altri agenti possono farlo. Consideriamo la formula:

J. R. Lucas non può asserire in modo consistente che questa formula è vera.

Se Lucas ne asserisse la verità, si starebbe contraddicendo, quindi Lucas non può asserirla in modo consistente, e quindi la formula dev'essere vera. Abbiamo così dimostrato che esiste una formula vera di cui Lucas non può asserire in modo consistente la verità, anche se altre persone (e macchine) possono farlo. Ma questo non intacca la nostra stima per Lucas.

In secondo luogo, il teorema di incompletezza di Gödel e i relativi risultati si applicano solo alla *matematica*, non ai *computer*. Nessuna entità – uomo o macchina che sia – può dimostrare cose che sono impossibili da dimostrare. Lucas e Penrose assumono falsamente che gli esseri umani possano aggirare tali limiti, come quando Lucas (1976) afferma: “Dobbiamo assumere di essere consistenti, affinché il nostro pensiero sia possibile”. Ma questa assunzione non va data per scontata: è noto che gli esseri umani sono inconsistenti. Questo

¹ Per esempio il balletto *Coppelia* (1970), il racconto *Do Androids Dream of Electric Sheep?* (1968), i film *AI* (2001), *Wall-E* (2008) e *Lei* (2013).

è certamente vero per il ragionamento della vita quotidiana, ma anche per un attento pensiero matematico. Un famoso esempio è quello della problema di colorare mappe con quattro colori: Alfred Kempe (1879) pubblicò una dimostrazione che fu ampiamente accettata per undici anni, finché Percy Heawood (1890) mise in luce un difetto.

In terzo luogo, il teorema di incompletezza di Gödel tecnicamente si applica soltanto a sistemi formali abbastanza potenti da gestire l'aritmetica. Questo include le macchine di Turing, e l'argomento di Lucas è in parte basato sull'asserzione che i computer siano equivalenti a macchine di Turing. Ma ciò non è del tutto vero. Le macchine di Turing sono infinite, mentre i computer (e i cervelli) sono finiti, e ogni computer può quindi essere descritto come un sistema (molto grande) in logica proposizionale che non è soggetto al teorema di incompletezza di Gödel. Lucas assume che gli esseri umani possano “cambiare idea” mentre i computer no, ma anche questo è falso: un computer può ritrattare una conclusione dopo nuove evidenze o ulteriori deliberazioni; può aggiornare il proprio hardware; può cambiare i suoi processi decisionali con l'apprendimento automatico o la riscrittura del software.

27.1.4 Misurare l'IA

test di Turing

Alan Turing, nel suo famoso articolo “Computing Machinery and Intelligence” (1950), suggerì che, invece di chiedersi se le macchine possano pensare, dovremmo chiederci se le macchine possano superare un test comportamentale che è poi diventato famoso come **test di Turing**. Il test richiede che un programma tenga una conversazione (attraverso messaggi di testo) con un esaminatore, per cinque minuti. L'esaminatore deve poi indovinare se ha conversato con un programma o una persona. Il programma supera il test se riesce a ingannare l'esaminatore per il 30% del tempo.² Il punto centrale, per Turing, non erano i dettagli precisi del test, ma l'idea di misurare l'intelligenza attraverso la prestazione in un certo tipo di attività comportamentale aperta, anziché mediante una speculazione filosofica.

Turing ipotizzò che entro l'anno 2000 un computer dotato di una memoria di archiviazione di un miliardo di unità sarebbe riuscito a superare il test; l'anno 2000 è ormai passato da un pezzo e ancora non siamo in grado di trovare un consenso uniforme sul fatto che qualche programma lo abbia superato o meno. Molte persone nel ruolo di esaminatori si sono fatte ingannare, quando non sapevano di conversare con un computer. Il programma ELIZA e chatbot Internet come MGONZ (Humphrys, 2008) e NATACHATA (Jonathan *et al.*, 2009) hanno ingannato ripetutamente i loro esaminatori, e il chatbot CYBERLOVER ha attirato l'attenzione delle forze di polizia a causa della sua inclinazione a convincere chi conversa con lui a divulgare informazioni personali sufficienti per un furto di identità.

Nel 2014 un chatbot denominato Eugene Goostman ingannò il 33% degli esaminatori amatoriali, privi di formazione specifica, in un test di Turing. Il programma sosteneva di essere un ragazzo ucraino con conoscenza limitata dell'inglese, e questo lo aiutò a spiegare i

² Quella riportata nel testo è una delle molte varianti del test di Turing. Il test originale proposto da Turing per sostituire la domanda “le macchine possono pensare?” è leggermente diverso e prende il nome di “gioco dell'imitazione” (imitation game). Il gioco è giocato da tre persone, un uomo (A), una donna (B) e un interrogante (C) che può essere di qualsiasi sesso. L'interrogante sta in una stanza separata da quelle degli altri due giocatori. Lo scopo del gioco per l'interrogante è determinare chi degli altri due sia l'uomo e chi la donna. L'interrogante può porre domande e ricevere risposte da A e B per mezzo di una telescrivente (tastiera e video). Lo scopo di A è ingannare l'interrogante per portarlo a una identificazione sbagliata. Lo scopo del gioco per B è di aiutare l'interrogante per portarlo a una identificazione corretta. A questo punto, Turing formula le domande: “cosa succede se una macchina prende il posto di A in questo gioco?” e “in questa nuova versione del gioco, l'interrogante sbaglierà con la stessa frequenza con la quale sbagliava nella versione originale giocata con un uomo e una donna?”. Secondo Turing, queste domande sostituiscono la domanda originale “le macchine possono pensare?” (N.d.C.).

suoi errori grammaticali. Forse il test di Turing in realtà è soltanto un test sulla credulità degli esseri umani. Finora nessun esaminatore bene addestrato è stato ingannato (Aaronson, 2014).

Le competizioni basate sul test di Turing hanno portato a sviluppare chatbot migliori, ma non hanno costituito un tema centrale di ricerca nella comunità dell'IA, che preferisce concentrarsi su competizioni basate su giochi come gli scacchi, Go o StarCraft II, o sul superare un esame di scienze delle scuole secondarie di primo grado, o sull'identificare oggetti nelle immagini. In molte di queste competizioni i programmi hanno raggiunto o superato le prestazioni di livello umano, ma questo non significa che siano simili agli umani al di là del compito specifico. Il punto è quello di migliorare la scienza di base e le tecnologie e di fornire strumenti utili, non di ingannare esaminatori.

27.2 Le macchine possono davvero pensare?

Secondo alcuni filosofi, una macchina che agisce in modo intelligente non sta *realmente* pensando, ma effettua solo una *simulazione* del pensiero. Tuttavia, la maggior parte dei ricercatori di IA non si preoccupa di tale distinzione, e lo scienziato Edsger Dijkstra (1984) disse: “Chiedersi se le *macchine possano pensare* ... è importante più o meno quanto chiedersi se i *sottomarini possano nuotare*”. La prima definizione dell’American Heritage Dictionary per il termine *nuotare* (*swim*) è: “Muoversi nell’acqua utilizzando arti, pinne o coda” e la maggior parte delle persone è d'accordo sul fatto che i sommergibili, essendo privi di arti, non possano nuotare. Il dizionario, inoltre, definisce *volare* (*fly*) come: “Muoversi nell’aria utilizzando ali o parti simili” e la maggior parte delle persone concorda sul fatto che gli aerei, dotati di parti simili ad ali, possono volare. Tuttavia, nessuna di queste domande, né le rispettive risposte, hanno alcuna rilevanza per la progettazione di aerei e sommergibili, ma si riferiscono semplicemente all’uso dei termini in linguaggio naturale (il fatto che le navi *nuotino* (“privet”) in lingua russa sottolinea ulteriormente il punto). Le persone di lingua inglese non hanno ancora trovato un accordo su una definizione precisa del verbo “pensare” (*think*): richiede un “cervello” o soltanto “parti simili a un cervello”?

Anche questo aspetto è stato affrontato da Turing, il quale osservò che non abbiamo mai *alcuna* evidenza diretta riguardo gli stati mentali interni di altri esseri umani, in una sorta di solipsismo mentale. Nondimeno, affermò Turing, “Anziché continuare ad argomentare su questo punto, solitamente si considera **per educata convenzione** che tutti pensino”. Turing sosteneva che avremmo dovuto estendere tale convenzione alle macchine, se avessimo potuto sperimentare macchine in grado di agire in modo intelligente. Tuttavia, oggi che abbiamo qualche esperienza al riguardo, sembra che la nostra disponibilità a considerare le macchine entità senzienti dipenda da un aspetto e da una voce simili a quelli umani almeno quanto dalla pura intelligenza.

per educata
convenzione

27.2.1 La stanza cinese

Il filosofo John Searle rifiuta l'educata convenzione. Il suo famoso argomento della **stanza cinese** (Searle, 1990) si svolge come segue: immaginate un essere umano, che capisce soltanto la lingua inglese, all'interno di una stanza che contiene un manuale di istruzioni scritto in inglese e varie pile di fogli di carta. Altri fogli di carta contenenti simboli indecifrabili vengono fatti entrare nella stanza passandoli sotto la porta. L'essere umano segue le istruzioni del manuale, trovando simboli nelle pile di fogli, scrivendo simboli su nuovi fogli di carta, riordinando le pile, e così via. Alla fine, le istruzioni indicheranno di trascrivere uno o più simboli su un foglio di carta da far passare sotto la porta verso il mondo esterno. Dall'esterno vediamo un sistema che riceve in input frasi in lingua cinese e genera risposte intelligenti e corrette in cinese.

stanza cinese

naturalismo biologico

Searle, quindi, argomenta: è dato il fatto che l'essere umano non conosce la lingua cinese. Il manuale di istruzioni e le pile di fogli sono soltanto oggetti di carta e quindi non conoscono il cinese. Perciò, non vi è conoscenza del cinese. E secondo Searle la stanza cinese fa la stessa cosa che farebbe un computer, perciò i computer non generano conoscenza.

Searle (1980) è un sostenitore del **naturalismo biologico**, secondo il quale gli stati mentali sono caratteristiche di alto livello emergenti, causate da processi fisici di basso livello *nei neuroni*, e sono le (non specificate) proprietà dei neuroni che contano: secondo i pregiudizi di Searle, i neuroni “sono capaci” e i transistor no. L'argomento di Searle ha trovato molte confutazioni, ma non il consenso. Lo stesso argomento potrebbe essere usato (forse da robot) per sostenere che un essere umano non possiede una reale conoscenza; dopo tutto, un essere umano è fatto di cellule, le cellule non hanno conoscenza, perciò non c'è conoscenza. In effetti è questo il tema del racconto di fantascienza di Terry Bisson (1990) *They're Made Out of Meat*, in cui robot alieni esplorano il pianeta Terra e non riescono a credere che degli ammassi di carne siano esseri senzienti, e come facciano rimane un mistero.

coscienza

qualia

27.2.2 Coscienza e qualia

In tutti i dibattiti relativi all'IA forte è coinvolto il tema della **coscienza**: la consapevolezza del mondo esterno e di sé, l'esperienza soggettiva del vivere. Il termine tecnico utilizzato per descrivere la natura intrinseca delle esperienze è **qualia** (parola latina che significa “qualità”). La grande domanda è se le macchine possano avere qualia. Nel film *2001 odissea nello spazio*, quando l'astronauta David Bowman sta scollegando i “circuiti cognitivi” del computer HAL 9000, quest'ultimo dice: *“Mi dispiace, Dave. Dave, la mia mente sta svanendo. Lo sento”*. HAL ha davvero dei sentimenti (e merita compassione)? O la risposta è soltanto un risultato algoritmico, non diverso da “Errore 404: non trovato”?

Una questione simile si pone anche per gli animali: i proprietari di animali da compagnia sono certi che i loro cani o gatti siano esseri coscienti, ma non tutti gli scienziati concordano. I grilli cambiano il loro comportamento in base alla temperatura, ma poche persone direbbero che questi animali sperimentino la *sensazione* di caldo o freddo.

Un motivo della difficoltà del problema della coscienza è che rimane mal definito anche dopo secoli di dibattito. Ma forse siamo vicini a una svolta. Recentemente alcuni filosofi hanno lavorato insieme a neuroscienziati, sotto gli auspici della Templeton Foundation, per avviare una serie di esperimenti che potrebbero risolvere alcune delle questioni aperte. I sostenitori delle due teorie principali relative al problema della coscienza, la teoria dello spazio di lavoro globale neuronale (*global neuronal workspace theory*) e la teoria dell'informazione integrata (*integrated information theory*), sono d'accordo sul fatto che questi esperimenti potrebbero confermare l'una o l'altra teoria – caso raro in campo filosofico.

Alan Turing (1950) concede che la questione della coscienza sia difficile, ma nega che essa abbia grande rilevanza per la pratica dell'IA: “Non vorrei dare l'impressione di pensare che non vi siano misteri relativi alla coscienza... Ma non penso che questi misteri debbano necessariamente essere risolti prima di poter rispondere alla domanda a cui siamo interessati in questo articolo”. Noi siamo d'accordo con Turing: ciò che ci interessa è creare programmi che si comportino in modo intelligente. Singoli aspetti della coscienza – consapevolezza, consapevolezza di sé, attenzione – possono essere programmati e andare a costituire parti di una macchina intelligente. Il progetto ulteriore di creare una macchina cosciente nello stesso modo in cui gli esseri umani lo sono non è alla nostra portata. Concordiamo sul fatto che per comportarsi in modo intelligente è necessario un certo grado di *consapevolezza*, che cambierà da un'attività all'altra, e che le attività che comportano l'interazione con gli esseri umani richiederanno un modello dell'esperienza soggettiva umana.

Per quanto riguarda la modellazione dell'esperienza, gli esseri umani hanno un chiaro vantaggio sulle macchine, perché possono usare il loro apparato soggettivo per apprezzare

l'esperienza soggettiva di altri. Per esempio, se volete sapere *che cosa si prova* quando qualcuno si dà una martellata sul pollice, potete darvi una martellata sul pollice. Le macchine non hanno questa capacità, anche se, a differenza degli esseri umani, possono eseguire il codice di altre macchine.

27.3 L'etica dell'IA

L'IA è una tecnologia potente, per cui abbiamo l'obbligo morale di usarla bene, promuovendone gli aspetti positivi ed evitando o mitigando quelli negativi.

Gli aspetti positivi sono molti. Per esempio, l'IA può salvare vite attraverso un miglioramento delle diagnosi mediche, nuove scoperte in medicina, una migliore previsione di eventi meteorologici estremi, rendendo più sicura la guida delle automobili con l'assistenza alla guida e (eventualmente) la guida autonoma. Offre anche molte opportunità di migliorare la nostra vita. Il programma AI for Humanitarian Action di Microsoft applica l'IA per facilitare la ripresa da catastrofi naturali, rispondere alle esigenze dei bambini, proteggere i rifugiati, promuovere i diritti umani. Il programma AI for Social Good di Google opera per la protezione delle foreste pluviali, la giurisprudenza in tema di diritti umani, il monitoraggio dell'inquinamento, la misurazione delle emissioni di combustibili fossili, attività di consulenza in situazioni di crisi, controllo e *fact checking* delle notizie, prevenzioni dei suicidi, riciclaggio e altri temi. Il Center for Data Science for Social Good dell'Università di Chicago applica l'apprendimento automatico a problemi nell'ambito della giustizia penale, dello sviluppo economico, della sanità pubblica, dell'energia e dell'ambiente.

Le applicazioni dell'IA alla gestione delle colture e alla produzione di alimenti aiutano a sfamare il mondo. L'ottimizzazione di processi aziendali mediante l'apprendimento automatico renderà le attività più produttive incrementando il benessere e favorendo l'occupazione. L'automazione può sostituire i compiti noiosi e pericolosi svolti da molti lavoratori, consentendo loro di concentrarsi su aspetti più interessanti. Le persone disabili possono trarre vantaggio dall'assistenza di tecnologie di IA per la vista, l'udito e la mobilità. La traduzione automatica consente già di comunicare tra persone di culture diverse. Le soluzioni software di IA hanno un costo marginale di produzione quasi nullo, perciò potrebbero favorire un accesso più democratico alle tecnologie avanzate (anche se altri aspetti del software potrebbero invece favorire un accentramento del potere).

Non possiamo comunque ignorare gli aspetti negativi. Molte tecnologie nuove hanno avuto **effetti collaterali negativi** non previsti: la fissione nucleare ha portato a Chernobyl e alla minaccia della distruzione globale; il motore a combustione interna ha portato all'inquinamento dell'aria, al riscaldamento globale e alla costruzione di strade distruggendo paesaggi. Altre tecnologie possono avere effetti negativi anche quando sono usate nel modo previsto, come il gas sarin, i fucili AR-15 e il marketing via telefono. L'automazione creerà ricchezza, ma nell'attuale situazione economica gran parte di tale ricchezza andrà a chi possiede i sistemi automatizzati, aumentando le diseguaglianze economiche. Questo potrebbe essere un fattore distruttivo per una società che funziona bene. Nei paesi in via di sviluppo, il percorso tradizionale che prevedeva la crescita attraverso produzioni a basso costo di beni destinati all'esportazione potrebbe essere interrotto, con l'adozione da parte dei paesi ricchi di impianti di produzione automatizzati a livello locale. Le nostre decisioni etiche e politiche determineranno il livello di diseguaglianza che l'IA potrà generare.

effetti collaterali negativi

Tutti gli ingegneri e gli scienziati devono affrontare considerazioni di carattere etico su progetti da intraprendere o meno, e su come rendere sicura e vantaggiosa l'attuazione di un progetto. Nel 2010 l'Engineering and Physical Sciences Research Council del Regno Unito ha tenuto una riunione per sviluppare un insieme di principi relativi alla robotica. Negli anni a seguire, altri enti pubblici, organizzazioni nonprofit e aziende hanno creato sistemi di prin-

cipi analoghi. Il punto è che ogni organizzazione che crea tecnologia di IA, e ogni persona all'interno di tali organizzazioni, ha la responsabilità di assicurarsi che la tecnologia contribuisca al bene, non al male. I principi citati più spesso sono i seguenti:

Garantire la sicurezza	Imporre la responsabilità (<i>accountability</i>)
Garantire l'equità	Sostenere i diritti e i valori umani
Rispettare la privacy	Riflettere la diversità/inclusione
Promuovere la collaborazione	Evitare la concentrazione del potere
Fornire trasparenza	Riconoscere le implicazioni legali/politiche
Limitare gli usi pericolosi dell'IA	Tenere conto delle conseguenze per l'occupazione

Noteate che molti di questi principi, come “garantire la sicurezza”, si applicano a tutti i sistemi software o hardware, non solo ai sistemi di IA. Diversi principi sono espressi in modo vago, risultando così di difficile misurazione o applicazione. Ciò si deve in parte al fatto che l'IA è un campo molto grande, con molti sottocampi, ognuno dei quali ha un diverso insieme di norme stabilitesi nel tempo e relazioni diverse tra gli sviluppatori di IA e i portatori di interessi (*stakeholder*). Mittelstadt (2019) suggerisce che ogni sottocampo dovrebbe sviluppare linee guida più specifiche e realizzabili, nonché dei casi di riferimento.

27.3.1 Armi letali autonome

L'ONU definisce come arma letale autonoma un'arma in grado di localizzare, selezionare e ingaggiare (cioè uccidere) obiettivi umani senza supervisione da parte dell'uomo. Vari tipi di armi soddisfano alcuni di questi criteri; per esempio, le mine antiuomo sono usate fin dal diciassettesimo secolo, e sono in grado di selezionare e ingaggiare obiettivi, limitatamente al grado di pressione esercitato o alla quantità di metallo presente, ma non possono uscire e localizzare gli obiettivi da sole (le mine antiuomo sono bandite dal Trattato di Ottawa). I missili guidati, in uso dagli anni 1940, possono inseguire gli obiettivi, ma dev'essere un uomo a puntarli nella direzione grosso modo corretta. Le armi a ingaggio automatico con sistema di puntamento radar sono usate per difendere le navi da guerra fin dagli anni 1970; sono pensate principalmente per distruggere missili in arrivo, ma possono anche attaccare aerei con equipaggio. Spesso si usa il termine “autonomo” per descrivere velivoli senza equipaggio o **droni**, ma nella maggior parte dei casi si tratta di armi che sono pilotate da remoto e richiedono l'intervento dell'uomo per fare fuoco.

Nel momento in cui scriviamo, diversi sistemi missilistici sembrano aver oltrepassato il confine verso la piena autonomia. Per esempio, il missile Harop di Israele è un drone da combattimento con un'apertura alare di oltre tre metri e una testata di circa 23 chili. Può cercare per sei ore, in un'area geografica specificata, qualsiasi obiettivo che soddisfi un dato criterio, e una volta individuato lo distrugge. Il criterio potrebbe essere “emette un segnale radar simile a quello di un radar antiaereo” o “assomiglia a un carrarmato”. Il produttore turco STM pubblicizza il suo quadricottero Kargu – che porta fino a 1,5 kg di esplosivo – affermando che è in grado di “Colpire autonomamente ... obiettivi selezionati da immagini ... tracciare obiettivi in movimento ... antiuomo ... con riconoscimento facciale”.

Le armi autonome sono state indicate come la “terza rivoluzione in campo bellico” dopo la polvere da sparo e le armi nucleari. Il loro potenziale militare è evidente. Per esempio, pochi esperti dubitano che un aereo da combattimento autonomo possa sconfiggere qualsiasi pilota umano. Aerei, carrarmati e sommergibili a guida autonoma possono essere più economici, più veloci, più manovrabili e avere un raggio d'azione superiore a quello delle loro controparti dotate di equipaggio.

A partire dal 2014, presso l'ONU a Ginevra si sono tenuti regolarmente dei dibattiti, sotto gli auspici della “Convenzione su certe armi convenzionali” (CCW, Convention on Certain Conventional Weapons), sul tema di mettere al bando le armi autonome letali. Al momento

in cui scriviamo, 30 nazioni, dalla Cina alla Città del Vaticano, hanno dichiarato il loro sostegno a un trattato internazionale, mentre altri Paesi, tra cui Israele, Russia, Corea del Sud e Stati Uniti, si oppongono alla messa al bando.

Il dibattito sulle armi autonome comprende aspetti legali, etici e pratici. Gli aspetti legali sono governati principalmente dal CCW, che richiede la possibilità di discriminare tra combattenti e non combattenti, il giudizio della necessità o meno di un attacco, la valutazione della proporzione tra il valore militare di un obiettivo e la possibilità di danni collaterali. La possibilità di soddisfare questi criteri è una questione tecnica la cui risposta cambierà sicuramente nel tempo. Al momento in cui scriviamo, la discriminazione tra combattenti e non combattenti appare possibile in alcune circostanze e migliorerà certamente in tempi rapidi, ma necessità e proporzionalità non sono criteri attuabili per ora: richiedono che le macchine effettuino giudizi soggettivi e legati alla situazione, molto più difficili da elaborare rispetto al compito relativamente semplice di cercare e ingaggiare potenziali obiettivi. Per questi motivi, sarebbe lecito usare armi autonome soltanto in circostanze in cui un operatore umano possa predire ragionevolmente che l'esecuzione della missione non porterà a individuare civili come obiettivi, o ad attacchi non necessari o sproporzionati. Questo significa che, per il momento, soltanto missioni molto ben definite e particolari potrebbero essere intraprese da armi autonome.

Dal punto di vista etico, alcuni considerano moralmente inaccettabile delegare a una macchina la decisione di uccidere esseri umani. Per esempio, l'ambasciatore tedesco a Ginevra ha affermato che “non accetterà che una decisione sulla vita o la morte sia presa unicamente da un sistema autonomo”, mentre il Giappone “non ha alcun piano di sviluppare robot senza interventi umani (*out of the loop*), che potrebbero essere in grado di commettere omicidi”. Il Generale Paul Selva, che all'epoca era il secondo ufficiale più alto in grado negli Stati Uniti, disse nel 2017: “Non penso che sia ragionevole mettere dei robot nella condizione di poter decidere se uccidere una vita umana”. Alla fine António Guterres, segretario generale dell'ONU, affermò nel 2019 che “le macchine dotate del potere e della discrezionalità di uccidere senza l'intervento dell'uomo sono politicamente inaccettabili, moralmente ripugnanti e dovrebbero essere proibite dalle leggi internazionali”.

Più di 140 ONG di oltre 60 paesi fanno parte della “Campagna per fermare i robot killer”, e una lettera aperta redatta nel 2015 dal Future of Life Institute è stata firmata da oltre 4.000 ricercatori in IA³ e 22.000 altri.

Contro questa posizione si potrebbe sostenere che, con il miglioramento della tecnologia, dovrebbe essere possibile sviluppare armi in grado di ridurre le perdite civili rispetto a quelle provocate da piloti o soldati umani (e c'è anche l'importante vantaggio che le armi autonome riducono la necessità di rischiare le vite di piloti e soldati umani). I sistemi autonomi non cedono a fatica, frustrazione, isteria, paura, rabbia o vendetta, e non devono “sparare prima di fare domande” (Arkin, 2015). Le armi teleguidate hanno ridotto i danni collaterali rispetto alle bombe prive di guida, e allo stesso modo ci si potrebbe attendere che le armi intelligenti possano migliorare ulteriormente la precisione degli attacchi (contro questa posizione si veda Benjamin [2013] per un'analisi degli incidenti di guerra causati dai droni). Questa, apparentemente, è la posizione degli Stati Uniti nell'ultima sessione di trattative a Ginevra.

In modo forse controintuitivo, gli Stati Uniti sono anche tra le poche nazioni le cui norme attualmente impediscono di usare armi autonome. La *road map* del 2011 del Dipartimento della difesa statunitense specifica: “Nel prossimo futuro, le decisioni sull'uso della forza [da parte di sistemi autonomi] e la scelta di quali obiettivi singoli ingaggiare con forza letale saranno mantenute sotto il controllo umano”. Questa politica si basa principalmente su un

³ Inclusi i due autori di questo libro.

aspetto concreto: i sistemi autonomi non sono abbastanza affidabili per poter essere incaricati di prendere decisioni in ambito militare.

L'aspetto dell'affidabilità apparve evidente il 26 settembre 1983, quando sullo schermo del computer di Stanislav Petrov, missilista sovietico, apparve l'allarme di un attacco missilistico in arrivo. In base al protocollo, Petrov avrebbe dovuto avviare un controattacco nucleare, ma invece sospettò che l'allarme fosse dovuto a un errore e lo trattò come tale. Aveva ragione, e così fu evitata (per poco) la terza guerra mondiale. Non sappiamo che cosa sarebbe successo se non fosse stato coinvolto alcun essere umano.

L'affidabilità è una preoccupazione molto seria per i comandanti militari, che conoscono bene la complessità delle situazioni di battaglia. I sistemi di apprendimento automatico che operano senza problemi in fase di addestramento, a volte offrono prestazioni scadenti quando sono impiegati sul campo. Cyber-attacchi contro le armi autonome potrebbero dare origine a incidenti di fuoco amico; la disconnessione dell'arma da ogni tipo di comunicazione (supponendo che non sia stata ancora compromessa del tutto) potrebbe evitare tale pericolo, ma non si potrebbe più riprendere il controllo dell'arma in caso di malfunzionamento.

Il problema pratico principale delle armi autonome è che sono armi di distruzione di massa scalabili, nel senso che la scala di un attacco che potrebbe essere portato è proporzionale alla quantità di hardware che ci si può permettere di mettere in campo. Un quadricottero di 5 centimetri di diametro può portare un carico esplosivo letale, e un milione di questi apparecchi possono stare in un normale container da nave. Proprio perché sono autonome, queste armi non necessitano di un milione di supervisori umani per fare il loro lavoro.

In quanto armi di distruzione di massa, le armi autonome scalabili hanno un vantaggio, per chi attacca, rispetto alle armi nucleari e ai bombardamenti a tappeto: lasciano intatte le proprietà e possono essere applicate in modo selettivo per eliminare soltanto coloro che potrebbero costituire una minaccia per una forza di occupazione. Potrebbero certamente essere usate per cancellare un intero gruppo etnico, o tutti gli aderenti a una particolare religione. In molte situazioni sarebbero anche impossibili da tracciare. Queste caratteristiche rendono queste armi particolarmente attraenti per organizzazioni non statali.

Queste considerazioni, in particolare quelle relative alle caratteristiche che avvantaggiano chi attacca, suggeriscono che le armi autonome porteranno a una riduzione della sicurezza globale e nazionale per tutti. La risposta razionale dei governi sembra essere quella di impegnarsi a discutere su come controllare le armi, anziché in una corsa agli armamenti.

Il processo di redigere un trattato non è privo di difficoltà. L'IA è una tecnologia **duale**: le tecnologie IA che hanno applicazioni di pace quali il controllo di un aereo, il tracciamento visuale, la costruzione di mappe, la navigazione e la pianificazione multiagente, possono facilmente essere applicate anche a scopi militari. È facile trasformare un quadricottero autonomo in un'arma, semplicemente armandolo con un esplosivo e indirizzandolo alla ricerca di un obiettivo. Per affrontare questi aspetti sarà necessario implementare con attenzione disciplinati regimi di conformità con cooperazione industriale, come si è già potuto vedere con alcuni successi riscossi dalla convenzione sulle armi chimiche.

duale

27.3.2 Sorveglianza, sicurezza e privacy

Nel 1976 Joseph Weizenbaum avvertì che la tecnologia di riconoscimento vocale automatico avrebbe potuto causare una diffusione incontrollata delle intercettazioni e rappresentava quindi una potenziale minaccia per le libertà civili. Oggi quella minaccia si è realizzata, infatti la maggior parte delle comunicazioni elettroniche è gestita attraverso server che possono essere tenuti sotto controllo, e le città sono piene di microfoni e videocamere in grado di identificare e di tracciare le persone in base alla voce, al viso e all'andatura. Attività di sorveglianza che in passato richiedevano risorse umane costose e scarse, oggi possono essere svolte su vasta scala dalle macchine.

Nel 2018 c'erano almeno 350 milioni di **videocamere di sorveglianza** in Cina e 70 milioni negli Stati Uniti. La Cina e altri paesi hanno iniziato a esportare tecnologie di sorveglianza verso i paesi meno evoluti tecnologicamente, alcuni dei quali noti per i maltrattamenti nei confronti dei loro cittadini e per opprimere in modo sproporzionato le comunità poste ai margini. Nel campo dell'IA gli **ingegneri** dovrebbero essere ben chiari su quali usi della sorveglianza siano compatibili con i diritti umani e rifiutare di lavorare ad applicazioni non compatibili.

**videocamera
di sorveglianza**

Dato che le nostre istituzioni operano sempre di più online, diventiamo sempre più vulnerabili al cyber-crimine (phishing, frodi della carta di credito, botnet, ransomware) e al cyber terrorismo (che comprende attacchi potenzialmente letali come l'interruzione di energia negli ospedali e nelle fabbriche, o il dirottamento di auto a guida autonoma). L'apprendimento automatico può essere uno strumento potente per entrambe le parti che si contrappongono nella battaglia per la **cyber-sicurezza**. Chi attacca può usare tecnologie di automazione per individuare i punti deboli e applicare l'apprendimento con rinforzo per tentativi di phishing e blackmailing automatizzato. Chi si difende utilizza l'apprendimento non supervisionato per individuare schemi anomali nel traffico in entrata (Chandola *et al.*, 2009; Malhotra *et al.*, 2015) e varie tecniche di apprendimento automatico per individuare tentativi di frode (Fawcett e Provost, 1997; Bolton e Hand, 2002). Poiché le tecniche di attacco si fanno sempre più sofisticate, tutti i tecnici, non soltanto gli esperti di sicurezza, hanno la responsabilità di progettare sistemi sicuri fin dal loro primo sviluppo. Una previsione (Kanal, 2017) indica in 100 miliardi di dollari il valore del mercato per le tecnologie di apprendimento automatico nel campo della cyber-sicurezza al 2021.

cyber-sicurezza

Dato che interagiamo con i computer per quantità di tempo sempre maggiori, governi e grandi aziende raccolgono maggiori quantità di dati su di noi. Coloro che raccolgono dati hanno la responsabilità morale e legale di custodirli con attenzione. Negli Stati Uniti lo Health Insurance Portability and Accountability Act (HIPAA) e il Family Educational Rights and Privacy Act (FERPA) proteggono la riservatezza dei dati medici e didattici. La normativa dell'Unione europea sulla protezione dei dati (General Data Protection Regulation, GDPR) obbliga le aziende a progettare i loro sistemi tenendo conto della protezione dei dati e richiede di ottenere il consenso degli utenti per qualsiasi attività di raccolta o elaborazione di dati.

Contrapposto al diritto alla riservatezza degli individui vi è il valore che la società ottiene dalla condivisione dei dati. Vogliamo essere in grado di fermare i terroristi senza opprimere il dissenso pacifico, e vogliamo poter curare le malattie senza compromettere il diritto di chiunque a mantenere la riservatezza sui propri dati sanitari. Una pratica fondamentale è la **de-identificazione**, con cui si eliminano le informazioni che consentono l'identificazione personale (come il nome e il codice fiscale o il numero della tessera sanitaria) in modo che i ricercatori in campo medico possano usare i dati per compiere progressi a favore del bene comune. Il problema è che i dati de-identificati e condivisi potrebbero essere re-identificati. Per esempio, se nei dati sono stati rimossi il nome, il codice fiscale e l'indirizzo di residenza, ma sono rimasti la data di nascita, il genere e il codice di avviamento postale, allora, come ha mostrato Latanya Sweeney (2000), l'87% della popolazione statunitense può essere re-identificato in modo univoco. A sostegno della sua tesi, Sweeney ha re-identificato i dati della cartella clinica del governatore del suo stato, quando era stato ricoverato in ospedale. Nella competizione **Netflix Prize**, si rilasciavano i record de-identificati con le valutazioni dei film, e i concorrenti dovevano sviluppare un algoritmo di apprendimento automatico in grado di predire con accuratezza quali film avrebbe gradito una singola persona. Ma i ricercatori furono in grado di re-identificare singoli utenti facendo corrispondere i dati di un voto contenuti nel database di Netflix con la data di una posizione simile nell'Internet Movie Database (IMDB), in cui gli utenti a volte usano i loro nomi veri (Narayanan e Shmatikov, 2006).

de-identificazione

Netflix Prize

Questo rischio può essere in qualche modo mitigato **generalizzando i campi**: per esempio, sostituendo la data di nascita esatta con il semplice anno di nascita, o un intervallo più ampio

k-anonimato

come “da 20 a 30 anni di età”. L'eliminazione totale di un campo può essere vista come una forma di generalizzazione a “qualsiasi”. Tuttavia, la generalizzazione da sola non garantisce che i record siano immuni dalla possibilità di re-identificazione: può essere che esista una sola persona che abita nell'area del codice di avviamento postale 94720 e ha un'età da 90 a 100 anni. Una proprietà utile è il **k-anonimato**: un database è *k*-anonimizzato se ogni record è indistinguibile da almeno $k - 1$ altri record. Se ci sono record più unici di questo, dovrebbero essere generalizzati ulteriormente.

interrogazione di dati aggregati

Un'alternativa alla condivisione di record de-identificati è quella di mantenere tutti i record privati, ma consentendo l'**interrogazione di dati aggregati**. Viene fornita un'API per interrogazioni sul database, e le interrogazioni (o query) valide ricevono una risposta che riepiloga i dati con un conteggio o una media. Tuttavia, non viene data alcuna risposta se vi è la possibilità di violare determinate garanzie di riservatezza. Per esempio, potremmo consentire a un epidemiologo di chiedere, per ciascun codice di avviamento postale, la percentuale di persone ammalate di cancro. Per codici di avviamento postale in cui risiedono almeno n persone verrebbe fornita una percentuale (con un po' di rumore casuale), mentre non verrebbe fornita alcuna risposta per codici i cui residenti sono meno di n .

Occorre prendere delle misure di protezione contro la possibilità di de-identificazione mediante interrogazioni multiple. Per esempio, se la query “salario medio e numero di dipendenti dell'azienda XYZ di età da 30 a 40 anni” fornisce la risposta [€81.234, 12] e la query “salario medio e numero di dipendenti dell'azienda XYZ di età da 30 a 41 anni” fornisce la risposta [€81.199, 13], e se utilizziamo LinkedIn per trovare l'unico 41enne dipendente dell'azienda XYZ, allora lo abbiamo identificato e possiamo calcolare il suo salario esatto, anche se tutte le risposte fornite riguardavano 12 o più persone. Il sistema deve essere progettato con cura in modo da offrire protezione contro questi attacchi, combinando limitazioni alle query eseguibili (per esempio, consentendo di interrogare soltanto un insieme predefinito di intervalli di età non sovrapponibili) e precisione dei risultati (per esempio con query che forniscono risposte come “circa €81.000”).

privacy differenziale

Una garanzia più forte è offerta dalla **privacy differenziale**, che assicura che un aggressore non possa usare delle interrogazioni per re-identificare qualsiasi individuo presente nel database, anche se può effettuare più interrogazioni e ha accesso a database separati ma collegati. La risposta alla query utilizza un algoritmo randomizzato che aggiunge una piccola quantità di rumore al risultato. Dato un database D , un record nel database r , una query Q e una possibile risposta y alla query, diciamo che il database D ha privacy ϵ -differenziale se la probabilità logaritmica della risposta y varia di meno di ϵ quando aggiungiamo il record r :

$$|\log P(Q(D) = y) - \log P(Q(D + r) = y)| \leq \epsilon.$$

In altre parole, il fatto che altre persone decidano di inserire i loro dati nel database non fa grande differenza per le risposte che chiunque può ottenere, perciò non vi è un disincentivo a inserire i dati per motivi di riservatezza. Molti database sono progettati in modo da garantire la privacy differenziale.

apprendimento federato

Finora abbiamo considerato il tema della condivisione di dati de-identificati da un database centrale. L'approccio dell'**apprendimento federato** (Konečný *et al.*, 2016), invece, non prevede un database centrale, ma gli utenti mantengono i loro database locali per mantenere privati i loro dati. Tuttavia, gli utenti possono condividere parametri di un modello di apprendimento automatico che viene potenziato con i loro dati, senza il rischio di rivelare alcun dato riservato. Immaginate un'applicazione di comprensione del parlato che gli utenti possono eseguire localmente sul loro telefono; l'applicazione contiene una rete neurale di base, che viene poi migliorata mediante una fase di addestramento locale sulle parole ascoltate attraverso il telefono dell'utente. Periodicamente, i proprietari dell'applicazione interrogano un sottoinsieme degli utenti chiedendo loro i valori dei parametri della loro rete locale migliorata, ma non i loro dati grezzi. I valori dei parametri vengono combinati tra loro per for-

mare un nuovo modello migliorato che viene poi reso disponibile a tutti gli utenti, affinché tutti possano sfruttare il vantaggio dell'addestramento effettuato dagli altri.

Affinché questo schema preservi la privacy, dobbiamo essere in grado di garantire che i parametri del modello condivisi da ciascun utente non possano essere oggetto di *reverse engineering*. Inviando i parametri grezzi, vi sarebbe la possibilità che un malintenzionato ispezionandoli possa dedurne se, per esempio, una certa parola è stata ascoltata attraverso il telefono dell'utente. Un modo per eliminare questo rischio è quello di utilizzare l'**aggregazione sicura** (Bonawitz *et al.*, 2017). L'idea è che il server centrale non deve conoscere il valore esatto del parametro da ogni utente distribuito; gli basta conoscere il valore medio per ciascun parametro su tutti gli utenti interrogati. In questo modo, ogni utente può camuffare i valori dei suoi parametri aggiungendo una maschera unica a ciascun valore; finché la somma delle maschere è zero, il server centrale sarà in grado di calcolare la media corretta. I dettagli di questo protocollo garantiscono che sia efficiente in termini di comunicazione (meno della metà dei bit trasmessi corrisponde alla mascheratura), robusto alla possibilità che singoli utenti non rispondano, e sicuro rispetto alle attenzioni di malintenzionati, intercettatori o anche di un server centrale antagonista.

aggregazione sicura

27.3.3 Equità e distorsione

L'apprendimento automatico sta potenziando e a volte rimpiazzando i processi decisionali degli esseri umani in situazioni importanti: a chi concedere un finanziamento, a quale quartiere destinare una pattuglia di polizia, a chi concedere il rilascio in attesa di giudizio o la libertà condizionale. Tuttavia, i modelli di apprendimento automatico possono perpetuare il **pregiudizio o distorsione sociale**. Consideriamo l'esempio di un algoritmo usato per prevedere se gli imputati in un procedimento penale tenderanno a commettere nuovamente reati e, quindi, se vadano rilasciati prima del giudizio. Un tale sistema potrebbe perpetuare pregiudizi e distorsioni razziali o di genere derivati dagli esempi forniti nell'insieme dei dati per l'addestramento. I progettisti di sistemi di apprendimento automatico hanno la responsabilità morale di assicurarsi che i loro sistemi siano equi. In settori regolamentati come quelli del credito, della formazione, dell'occupazione e dell'immobiliare, c'è anche una responsabilità legale. Ma che cos'è l'equità? Per definirla si possono utilizzare diversi criteri; di seguito ne elenchiamo sei dei più comuni.

distorsione sociale

- **Equità individuale:** il requisito che gli individui siano trattati in modo simile ai loro simili, a prescindere dalla loro classe sociale.
- **Equità di gruppo:** il requisito che due classi siano trattate in modo simile, secondo indicatori misurati mediante statistiche di riepilogo.
- **Equità ottenuta attraverso l'inconsapevolezza:** se eliminiamo dal dataset gli attributi di razza e di genere, potremmo pensare che il sistema non possa fare discriminazioni basate su tali attributi. Sfortunatamente, però, sappiamo che i modelli di apprendimento automatico sono in grado di predire variabili latenti (come razza e genere) date altre variabili correlate (come codice di avviamento postale e occupazione). Inoltre, se si eliminano questi attributi risulterà impossibile verificare che vi siano pari opportunità o pari esiti. Nonostante ciò, alcuni paesi (come la Germania) hanno scelto questo approccio per le loro statistiche demografiche (a prescindere dall'utilizzo di modelli di apprendimento automatico).
- **Equo esito:** il concetto che ogni classe demografica ottenga gli stessi risultati; deve esserci **parità demografica**. Per esempio, supponiamo di dover decidere se approvare richieste di finanziamento. L'obiettivo è quello di approvare le richieste di persone che restituiranno il prestito e non approvare quelle di persone che non riusciranno a restituirlo. La parità demografica afferma che maschi e femmine dovrebbero ottenere l'approvazione dei finanziamenti con le stesse percentuali di successo. Notate che questo è un criterio di

parità demografica

equità di gruppo che non garantisce affatto l'equità individuale: un richiedente ben qualificato potrebbe vedere la sua domanda respinta e un altro meno qualificato potrebbe vederla approvata, purché le percentuali complessive siano le stesse. Inoltre, questo approccio favorisce la correzione di passate distorsioni rispetto alla precisione della predizione. Se un uomo e una donna sono uguali da tutti i punti di vista, salvo che la donna riceve una retribuzione inferiore per lo stesso lavoro, la domanda di finanziamento della donna dovrebbe essere approvata, perché la donna sarebbe uguale all'uomo se non fosse per distorsioni del passato, o dovrebbe essere respinta perché un salario inferiore costituisce in effetti un fattore che predisponde all'incapacità di rimborsare il prestito?

- **Pari opportunità:** l'idea che le persone che sono realmente in grado di rimborsare un prestito debbano avere un'equa possibilità di essere correttamente classificate come tali a prescindere dal genere. Questo approccio è anche detto "bilanciato" e può generare esiti non equi, inoltre ignora l'effetto della distorsione nei processi sociali che hanno prodotto i dati utilizzati per l'addestramento.
- **Pari impatto:** le persone che hanno probabilità simili di rimborsare il finanziamento dovrebbero avere la stessa utilità attesa a prescindere dalla classe a cui appartengono. Questo va oltre le pari opportunità, nel senso che considera sia i benefici di una predizione vera, sia i costi di una predizione falsa.

Esaminiamo ora come entrano in gioco questi aspetti in un particolare contesto. COMPAS è un sistema commerciale per la valutazione della possibilità di recidiva. Assegna a un imputato di un processo penale un **punteggio di rischio** che viene poi usato da un giudice come ausilio alle decisioni: è sicuro rilasciare l'imputato prima del giudizio, o è meglio mantenerlo in custodia cautelare? In caso di condanna, quanto tempo di reclusione dovrebbe prevedere la sentenza? Va concessa la libertà condizionale? Vista l'importanza di queste decisioni, il sistema è stato oggetto di esame approfondito (Dressel e Farid, 2018).

ben calibrato

COMPAS è progettato per essere **ben calibrato**: tutti gli individui a cui l'algoritmo assegna lo stesso punteggio dovrebbero avere approssimativamente la stessa probabilità di recidiva, a prescindere dalla razza. Per esempio, tra tutte le persone a cui il modello assegna un punteggio di rischio di 7 su 10, il 60% dei bianchi e il 61% dei neri recidivano. I progettisti, quindi, affermano che il sistema raggiunge l'obiettivo dell'equità.

D'altra parte, COMPAS non raggiunge l'obiettivo delle pari opportunità: la percentuale di coloro che non recidiva ma è stato erroneamente valutato ad alto rischio è risultata del 45% per i neri e del 23% per i bianchi. Nel caso *Stato contro Loomis*, in cui un giudice ha utilizzato COMPAS per determinare la sentenza per l'imputato, Loomis ha sostenuto che il meccanismo interno e segreto dell'algoritmo ha violato i suoi diritti processuali. In questo caso la corte suprema del Wisconsin ha determinato che la sentenza non sarebbe stata diversa se non si fosse utilizzato COMPAS, tuttavia ha messo in guardia contro i problemi di accuratezza dell'algoritmo e i rischi per gli imputati appartenenti a minoranze. Altri ricercatori hanno messo in discussione l'appropriatezza di utilizzare algoritmi in applicazioni come la determinazione di sentenze in tribunale.

Potremmo sperare in un algoritmo che sia allo stesso tempo ben calibrato e offra pari opportunità, ma come dimostrano Kleinberg *et al.* (2016), questo è impossibile. Se le classi di base sono diverse, qualsiasi algoritmo che sia ben calibrato non fornirà pari opportunità, e vice versa. Come possiamo pesare i due criteri? Una possibilità è quella di considerare un pari impatto. Nel caso di COMPAS, questo significa pesare l'utilità negativa della possibilità che alcuni imputati siano erroneamente classificati ad alto rischio perdendo così la loro libertà, rispetto al costo per la società di un crimine commesso in più, e trovare il punto che ottimizza il compromesso. La procedura è complessa, perché è necessario considerare diversi costi. Esistono costi a livello individuale – un imputato che viene recluso per errore soffre un costo, come la vittima di un imputato che viene rilasciato e poi recidiva. Ma oltre a questi ci sono costi

a livello di gruppo – tutti hanno una certa paura di essere portati in prigione per errore, o di essere vittima di un crimine, e tutti i contribuenti partecipano ai costi della gestione di penitenziari e tribunali. Se diamo un valore a queste paure e a questi costi in proporzione alla dimensione dei gruppi, rischiamo che l'utilità per la maggioranza sia raggiunta a spese di una minoranza.

Un altro problema generale relativo alla valutazione del recidivismo, a prescindere dal modello usato, è che non disponiamo di dati provenienti da osservazioni dirette e non distorte. I dati non ci dicono chi ha *commesso* un crimine, tutto ciò che sappiamo è chi è stato *condannato* per un crimine. Se la polizia che effettua l'arresto, il giudice o la giuria sono distorti, allora i dati saranno parziali o distorti. Se un maggior numero di poliziotti tiene sotto controllo alcuni quartieri, allora i dati saranno distorti nei confronti dei residenti in tali quartieri. Soltanto gli imputati che vengono rilasciati sono candidati a recidivare, perciò se i giudici che prendono le decisioni di rilascio sono distorti, i dati potrebbero essere distorti. Se si ipotizza che i dati siano distorti a causa del fatto che i dati non distorti, pur esistenti, siano stati corrotti da un agente distorto, allora esistono tecniche per recuperare un'approssimazione dei dati non distorti. Jiang e Nachum (2019) descrivono diversi scenari e le tecniche relative.

Un altro rischio è quello che le tecniche di apprendimento automatico possano essere usate per *giustificare* le distorsioni. Se le decisioni sono prese da un essere umano distorto dopo aver consultato un sistema di apprendimento automatico, l'umano può dire: "Ecco come la mia interpretazione del modello supporta la mia decisione, che quindi non dovreste mettere in discussione". Ma altre interpretazioni potrebbero portare a una decisione opposta.

A volte l'equità impone di riconsiderare la funzione obiettivo, non i dati o l'algoritmo. Per esempio, quando si prendono decisioni in tema di assunzioni per un impiego, se l'obiettivo è quello di assumere i candidati più qualificati, rischiamo di premiare in modo iniquo coloro che hanno avuto vantaggi nelle opportunità di formazione durante le loro vite, andando così a rafforzare il classismo. Se invece l'obiettivo è quello di assumere i candidati più capaci di imparare sul lavoro, abbiamo maggiori possibilità di attraversare i confini di classe e scegliere da un bacino più ampio. Molte aziende dispongono di programmi progettati proprio per selezionare quei tipi di persone, e risulta che dopo un anno di addestramento i dipendenti assunti in questo modo ottengano prestazioni pari a quelle dei candidati scelti con i metodi tradizionali. Similmente, soltanto il 18% dei laureati in informatica negli Stati Uniti sono donne, ma alcune istituzioni scolastiche, come la Harvey Mudd University, hanno ottenuto la parità al 50% utilizzando un approccio orientato sull'incoraggiare e assistere coloro che iniziano il corso di laurea in informatica, ponendo una particolare attenzione a coloro che iniziano con meno esperienza di programmazione.

Un'altra complicazione è quella di decidere quali classi meritino protezione. Negli Stati Uniti il Fair Housing Act ha riconosciuto sette classi da proteggere: razza, colore, religione, paese di origine, genere, disabilità e stato di famiglia. Altre leggi locali, statali e federali riconoscono altre classi, tra cui orientamento sessuale e gravidanza, stato civile e condizione di veterano militare. È equo il fatto che queste classi contino per alcune leggi e non per altre? La legge internazionale sui diritti umani, che comprende un ampio insieme di classi protette, è un quadro di riferimento che può favorire l'armonizzazione delle tutele attraverso i vari gruppi.

Anche in assenza di distorsione sociale, la **disparità della dimensione del campione** può condurre a risultati distorti. La maggior parte dei data set contiene meno dati di addestramento riferiti a individui appartenenti alle minoranze rispetto a quelli delle maggioranze. Gli algoritmi di apprendimento automatico offrono una maggiore accuratezza disponendo di più dati di addestramento, perciò questa disparità nei dati significa che per i membri delle classi minoritarie l'accuratezza sarà minore. Per esempio, Buolamwini e Gebru (2018) hanno esaminato un servizio di identificazione del genere basato su un sistema di visione artificiale, determinando che il servizio otteneva una precisione quasi perfetta per i maschi di pelle

**disparità
della dimensione
del campione**

chiara e un tasso di errore del 33% per le femmine di pelle scura. Un modello vincolato potrebbe non essere in grado di gestire contemporaneamente in modo adeguato le classi di minoranza e di maggioranza – un modello di regressione lineare potrebbe minimizzare l'errore medio eseguendo un fitting solo per la classe maggioritaria, e in un modello SVM i vettori di supporto potrebbero corrispondere tutti a membri della classe maggioritaria.

Distorsioni possono essere introdotte anche nel processo di sviluppo del software (a prescindere dal fatto che questo utilizzi l'apprendimento automatico). I tecnici che effettuano il debugging di un sistema tendono a notare più spesso e a correggere i problemi che li riguardano maggiormente. Per esempio, è difficile notare che un'interfaccia utente non funziona per i daltonici, se non si è daltonici, o che una traduzione in lingua Urdu è sbagliata, se non si conosce la lingua Urdu.

Come possiamo difenderci da tutte queste distorsioni? Per prima cosa occorre capire i limiti dei dati che si utilizzano. Alcuni studi hanno suggerito che i data set (Gebru *et al.*, 2018; Hind *et al.*, 2018) e i modelli (Mitchell *et al.*, 2019) dovrebbero essere forniti di annotazioni: dichiarazioni di provenienza, sicurezza, conformità e idoneità all'uso. La strategia è simile a quella dei **data sheet** o schede tecniche che accompagnano i componenti elettronici, come i resistori: consentono ai progettisti di decidere quali componenti usare. Oltre ai data sheet, è importante addestrare i tecnici a tenere conto di equità e distorsioni, sia a scuola, sia nella formazione lavorativa. La presenza di una diversità di tecnici con formazioni differenti facilita il compito di notare la presenza di problemi nei dati o nei modelli. Uno studio dell'AI Now Institute (West *et al.*, 2019) ha determinato che soltanto il 18% degli autori che partecipano alle più importanti conferenze sull'IA e soltanto il 20% dei docenti di IA sono donne. I neri che lavorano nell'IA sono meno del 4%. I tassi rilevati nei laboratori di ricerca del settore sono simili. La diversità si potrebbe aumentare attuando programmi specifici più a monte nella filiera dell'istruzione – all'università o nella scuola superiore – e con una maggiore consapevolezza a livello professionale. Joy Buolamwini ha finanziato l'Algorithmic Justice League per aumentare la consapevolezza su questo tema e sviluppare buone pratiche di responsabilizzazione.

Un'altra idea è quella di rimuovere le distorsioni nei dati (Zemel *et al.*, 2013). Si potrebbe per esempio aumentare il campionamento dalle classi minoritarie per compensare la disparità nella dimensione del campione. Tecniche come SMOTE, il sovraccampionamento sintetico sulla minoranza (Chawla *et al.*, 2002) o ADASYN, l'approccio di campionamento sintetico adattativo per l'apprendimento sbilanciato (He *et al.*, 2008), offrono metodi di sovraccampionamento strutturati. Potremmo esaminare la provenienza dei dati e, per esempio, eliminare gli esempi relativi a giudici che hanno mostrato distorsioni o pregiudizi nei loro casi passati. Alcuni analisti sono contrari all'idea di scartare dati e consigliano invece di costruire un modello gerarchico dei dati che includa le fonti di distorsione, in modo da poterle modellare e compensare. Google e NeurIPS hanno cercato di diffondere la consapevolezza di questo problema sponsorizzando l'Inclusive Images Competition, in cui i concorrenti addestrano una rete utilizzando un data set di immagini etichettate raccolte nel Nord America e in Europa, e poi lo testano su immagini prese da ogni parte del mondo. Il problema è che, con questo data set, è facile applicare l'etichetta "sposa" a una donna che porta un abito da matrimonio secondo gli standard occidentali, mentre è più difficile riconoscere gli abiti matrimoniali delle tradizioni africane e indiane.

Un'altra idea è quella di inventare nuovi modelli e algoritmi di apprendimento automatico che siano più resistenti alle distorsioni; e l'idea finale è quella di lasciare che un sistema effettui raccomandazioni iniziali distorte, ma poi addestrare un secondo sistema a rimuovere le distorsioni del primo. Bellamy *et al.* (2018) hanno introdotto il sistema IBM AI FAIRNESS 360, che fornisce una struttura per tutte queste idee. Prevediamo che in futuro l'uso di strumenti simili a questo aumenterà.

data sheet

Come potete assicurarvi che i sistemi che costruirete siano equi? Sta emergendo un insieme di buone pratiche (che tuttavia non vengono sempre seguite), descritte di seguito.

- Assicuratevi che gli ingegneri software parlino con specialisti di scienze sociali ed esperti del campo per capire i problemi e i punti di vista, e prendano in considerazione fin dall'inizio il tema dell'equità.
- Create un ambiente che favorisca lo sviluppo di una squadra di tecnici diversificata e che possa rappresentare la società nel suo complesso.
- Definite quali gruppi il vostro sistema supporterà: persone che parlano lingue diverse, persone di età differenti, persone con diversi livelli di vista e udito, e così via.
- Ottimizzate per una funzione obiettivo che incorpori l'equità.
- Esamine i dati cercando distorsioni e correlazioni tra attributi protetti e altri attributi.
- Cercate di capire come sia effettuata qualsiasi annotazione umana dei dati, fissate obiettivi da raggiungere per l'accuratezza delle annotazioni e verificate che tali obiettivi siano raggiunti.
- Non limitatevi a tracciare metriche generali per il vostro sistema; assicuratevi di tracciare metriche per sottogruppi che potrebbero essere vittime di distorsioni.
- Include test del sistema che riflettano l'esperienza di gruppi minoritari di utenti.
- Impostate un meccanismo di retroazione (*feedback*) in modo che, ove si presentino problemi di equità, siano subito affrontati.

27.3.4 Fiducia e trasparenza

Rendere un sistema di IA accurato, equo e sicuro è una sfida, ma un'altra sfida è quella di convincere tutti di averlo fatto. Le persone devono potersi **fidare** dei sistemi che usano. Un sondaggio condotto da PwC nel 2017 ha rilevato che il 76% delle aziende rallentava l'adozione dell'IA per questioni di fiducia. Nel Paragrafo 19.9.4 del Volume 2 sono trattati alcuni degli approcci tecnici al problema della fiducia; qui di seguito discutiamo gli aspetti di tipo politico.

fiducia

Per guadagnarsi fiducia, ogni sistema deve passare attraverso un processo di **verifica e validazione** (V&V). Verifica significa controllare che il prodotto soddisfi le specifiche. Validazione significa assicurarsi che le specifiche corrispondano realmente alle esigenze dell'utente e delle altre parti coinvolte. Disponiamo di un'elaborata metodologia di V&V per l'ingegneria in generale, e per il software tradizionale sviluppato da esseri umani; per buona parte tale metodologia può essere applicata anche ai sistemi di IA. Tuttavia, i sistemi di apprendimento automatico hanno delle peculiarità e richiedono un processo V&V diverso, che non è ancora stato del tutto sviluppato. Abbiamo bisogno di verificare i dati da cui questi sistemi apprendono; dobbiamo verificare l'accuratezza e l'equità dei risultati anche di fronte a fattori incerti che rendono un risultato esatto impossibile da raggiungere; e dobbiamo inoltre verificare che altri non possano influenzare indebitamente il modello, né sottrarre informazioni attraverso apposite interrogazioni.

verifica e validazione

Uno strumento di fiducia è la **certificazione**; per esempio, l'organizzazione Underwriters Laboratories (UL) fu fondata nel 1894, in un'epoca in cui i consumatori temevano i rischi dell'energia elettrica. La certificazione UL degli elettrodomestici dava ai consumatori maggiore fiducia, e oggi UL sta prendendo in considerazione l'idea di entrare nel business dei test e delle certificazioni di prodotti per l'IA.

certificazione

Altri settori dispongono da lungo tempo di standard di sicurezza. Per esempio, l'ISO 26262 è uno standard internazionale per la sicurezza delle automobili, che descrive come sviluppare, produrre, operare e manutenere i veicoli in modo sicuro. Il settore dell'IA non è ancora giunto a questo livello di chiarezza, anche se si sta lavorando su alcuni approcci, come l'IEEE P7001, uno standard che definisce il design etico per sistemi autonomi e di IA

(Bryson e Winfield, 2017). È in corso un dibattito su quale tipo di certificazione sia necessario, e fino a che punto dovrebbe essere implementato dai governi, da organizzazioni professionali come l'IEEE, da certificatori indipendenti come UL, o attraverso l'autoregolamentazione dei produttori.

trasparenza

Un altro aspetto della fiducia è la **trasparenza**: i consumatori vogliono sapere che cosa accade all'interno di un sistema, e sapere che il sistema non lavora contro di loro per cattiva intenzione, un bug indesiderato o una distorsione sociale pervasiva che è stata assorbita dal sistema. In alcuni casi il sistema è trasparente nei confronti del consumatore, mentre in altri ci sono motivi legati alla proprietà intellettuale che portano a mantenere alcuni aspetti del sistema nascosti ai consumatori, ma visibili agli enti di regolamentazione e a quelli di certificazione.

IA spiegabile

Quando un sistema di IA respinge una richiesta di finanziamento, la persona che ha effettuato tale richiesta merita una spiegazione. In Europa è il Regolamento generale per la protezione dei dati (GDPR) che provvede a questo. Un sistema di IA che è in grado di spiegare se stesso è detto **IA spiegabile** (XAI, da *explainable AI*). Una buona spiegazione ha diverse proprietà: dev'essere comprensibile e convincente per l'utente, riflettere accuratamente il meccanismo di ragionamento del sistema, essere completa e specifica, nel senso che utenti diversi con condizioni o risultati diversi dovrebbero ottenere spiegazioni diverse.

È abbastanza facile fornire a un algoritmo decisionale l'accesso ai suoi stessi processi deliberativi, semplicemente registrandoli e rendendoli disponibili come strutture dati. Questo significa che le macchine alla fine potrebbero essere in grado di spiegare le loro decisioni meglio degli esseri umani. Inoltre, possiamo intraprendere passi opportuni per certificare che le spiegazioni fornite dalla macchina non siano ingannevoli (intenzionalmente o per un meccanismo di autoinganno), cosa che risulta più difficile con un essere umano.

Una spiegazione è un ingrediente utile ma non sufficiente per ottenere la fiducia. Un problema è che le spiegazioni non sono decisioni, ma racconti di decisioni. Come è mostrato nel Paragrafo 19.9.4 del Volume 2, diciamo che un sistema è interpretabile se possiamo ispezionare il codice sorgente del modello e vedere come funziona, e diciamo che un sistema è spiegabile se possiamo costruire una storia su come il sistema funziona anche se il sistema in sé è una scatola nera non interpretabile. Per spiegare una scatola nera non interpretabile dobbiamo costruire un sistema di spiegazione separato, effettuare il debugging e collaudarlo assicurandoci che sia sincronizzato con quello originale. E poiché gli esseri umani amano i bei racconti, siamo tutti troppo disponibili a farci trascinare da una spiegazione che appare buona. Prendete qualsiasi controversia politica attuale e riuscirete sempre a trovare due cosiddetti esperti che propongono spiegazioni diametralmente opposte, ma entrambe internamente consistenti.

audit

Un ultimo problema è che una spiegazione relativa a un singolo caso non fornisce un riepilogo di altri casi. Se la banca spiega: "Siamo spiacenti, lei non ha ottenuto il finanziamento perché ha un passato che presenta problemi finanziari", non sapete se tale spiegazione è accurata o se la banca ha una distorsione segreta nei vostri confronti, per qualche motivo. In questo caso avreste bisogno non di una semplice spiegazione, ma anche di un **audit** delle decisioni passate, con statistiche aggregate su vari gruppi demografici, per verificare se i tassi di approvazione dei finanziamenti della banca sono equilibrati.

La trasparenza implica anche sapere se si sta interagendo con un sistema di IA o con un essere umano. Toby Walsh (2015) ha sostenuto che "un sistema autonomo dovrebbe essere progettato in modo che sia difficile scambiarlo per qualcos'altro, e dovrebbe presentarsi come tale all'inizio di qualsiasi interazione", chiamando questa proposta "bandiera rossa" in onore del Locomotive Act del 1865, con cui nel Regno Unito si imponeva che in ogni veicolo a motore dovesse essere preceduto da una persona con una bandiera rossa per segnalare il pericolo in arrivo.

Nel 2019 la California ha approvato una legge che stabilisce: “È illecito per qualunque persona utilizzare un bot per comunicare o interagire con altre persone online, in California, con l'intento di nascondere all'interlocutore l'identità artificiale del sistema”.

27.3.5 Il futuro del lavoro

Dalla prima rivoluzione agricola (10.000 A.C.) alla rivoluzione industriale (verso la fine del diciottesimo secolo), fino alla rivoluzione verde nella produzione agricola (anni 1950), le nuove tecnologie hanno cambiato il modo in cui l'umanità vive e lavora. Una preoccupazione fondamentale dovuta all'avanzare dell'IA è che il lavoro umano diventi obsoleto. Aristotele, nel Libro I della sua opera *Politica*, presenta il punto principale in modo molto chiaro:

Se ogni strumento potesse svolgere il suo lavoro da solo obbedendo o anticipando i comandi di altri ... se, in modo simile, la spola tessesse e il pletto toccasse la lira senza una mano a guidarli, allora i capi non avrebbero bisogno di servi, né i padroni di schiavi.

Tutti concordano con l'osservazione di Aristotele che vi è un'immediata riduzione dell'occupazione quando un datore di lavoro trova un metodo meccanico in grado di svolgere il lavoro che in precedenza era svolto da una persona. Il problema è se i cosiddetti effetti di compensazione che seguono – e che tendono ad aumentare l'occupazione – riusciranno a pareggiare tale riduzione. Il principale effetto di compensazione è l'aumento della ricchezza complessiva dovuto alla maggiore produttività, che a sua volta implica una maggiore domanda di beni e tende ad aumentare l'occupazione. Per esempio, PwC (Rao e Verweij, 2017) predice che nel 2030 l'IA contribuirà per 15 mila miliardi di dollari annui al PIL globale. I settori della sanità e dei trasporti dovrebbero ottenere i maggiori guadagni nel breve termine. Tuttavia, i vantaggi dell'automazione non si sono ancora trasferiti nella nostra economia: l'attuale tasso di crescita della produttività del lavoro è inferiore ai livelli standard del passato. Brynjolfsson *et al.* (2018) tentano di spiegare questo paradosso suggerendo che il tempo che passa tra lo sviluppo di tecnologia di base e la sua implementazione nell'economia è più lungo di quanto si creda.

In passato le innovazioni tecnologiche hanno sempre causato l'esclusione di alcune persone dal lavoro. Le tessitrici sono state rimpiazzate dai telai automatizzati negli anni 1810, causando le proteste dei luddisti; costoro non erano contro la tecnologia *in sé*, volevano solo che le macchine fossero usate da lavoratori capaci e retribuiti con un buon salario per produrre beni di alta qualità, invece che da lavoratori poco competenti per produrre beni di scarsa qualità a salari inferiori. La distruzione globale dei posti di lavoro avvenuta durante gli anni 1930 portò John Maynard Keynes a coniare il termine **disoccupazione tecnologica**. In entrambi i casi citati, e in molti altri, i livelli di occupazione alla fine sono ripresi.

**disoccupazione
tecnologica**

Durante la maggior parte del ventesimo secolo, la visione economica prevalente era che la disoccupazione tecnologica fosse al più un fenomeno di breve termine. L'aumento della produttività avrebbe sempre portato a un aumento della ricchezza e della domanda, e quindi alla crescita netta del lavoro. Un esempio comune è quello degli impiegati di banca: benché gli sportelli automatici abbiano sostituito gli esseri umani nel lavoro di contare i contanti per le operazioni di prelievo, hanno anche reso meno costosa la gestione di una filiale di banca, perciò il numero delle filiali è aumentato, con un aumento del numero di impiegati nel settore. Anche la natura del lavoro è cambiata, con una riduzione delle attività di routine e la richiesta di competenze economiche più avanzate. L'effetto netto dell'automazione sembra essere quello di eliminare *attività* più che *lavori*.

La maggioranza dei commentatori prevede che lo stesso accadrà con la tecnologia dell'IA, almeno nel breve termine. Gartner, McKinsey, Forbes, il World Economic Forum e il Pew Research Center hanno pubblicato nel 2018 report che prevedevano un saldo netto positivo dell'occupazione a causa dell'introduzione di tecnologie di automazione basate sull'IA. Alcuni analisti, tuttavia, ritengono che questa volta le cose andranno diversamente. Nel 2019

IBM ha predetto che 120 milioni di persone avrebbero perso il lavoro a causa dell'automazione entro il 2022, e Oxford Economics ha predetto che 20 milioni di posti di lavoro nel settore manifatturiero rischiano di andare perduti a causa dell'automazione entro il 2030.

Frey e Osborne (2017) hanno studiato 702 diversi tipi di lavori, stimando che il 47% di essi rischiano di essere automatizzati, nel senso che almeno alcune delle attività coinvolte in quei tipi di lavori possono essere svolte dalle macchine. Per esempio, quasi il 3% della forza lavoro negli Stati Uniti è rappresentato da autisti, e in alcuni distretti, fino al 15% dei lavoratori maschi sono autisti. Come è mostrato nel Capitolo 26 del Volume 2, l'attività di guidare potrebbe essere eliminata dall'introduzione di auto/camion/autobus/taxi a guida autonoma.

È importante distinguere tra le occupazioni e le attività svolte in tali occupazioni. McKinsey stima che soltanto il 5% delle occupazioni sia completamente automatizzabile, ma che per il 60% delle occupazioni si possa automatizzare circa il 30% delle attività svolte. Per esempio, nel futuro gli autisti di camion passeranno meno tempo tenendo in mano il volante e più tempo ad assicurarsi che i beni siano prelevati e consegnati in modo appropriato, a svolgere funzioni di servizio clienti e vendite alla partenza e all'arrivo, e magari a gestire convogli di tre o più camion robotizzati. La sostituzione di tre autisti con un unico gestore di convoglio implica una perdita netta di posti di lavoro, ma se i costi del trasporto diminuiscono, aumenterà la domanda, che farà riguadagnare alcuni posti di lavoro – anche se forse non tutti. Un altro esempio: nonostante i molti progressi compiuti nell'applicazione di sistemi di apprendimento automatico alla radiologia medica, finora i radiologi sono stati arricchiti, non sostituiti, da questi strumenti. Alla fine occorre fare una scelta su come usare l'automazione: vogliamo focalizzarci sul *taglio dei costi*, e quindi considerare la perdita di un posto di lavoro come un dato positivo, oppure vogliamo concentrarci sul *miglioramento della qualità*, rendendo migliore la vita dei lavoratori e dei clienti?

È difficile prevedere i tempi esatti in cui avranno luogo le automazioni, ma attualmente, e per i prossimi anni a venire, al centro dell'attenzione è l'automazione di attività di analisi strutturate, come la lettura di immagini radiografiche, la gestione delle relazioni con la clientela (per esempio, bot che filtrano automaticamente le lamentele dei clienti e sono in grado di rispondere fornendo suggerimenti) e l'**automazione dei processi aziendali**, che combina documenti testuali e dati strutturati per prendere decisioni aziendali e migliorare i flussi di lavoro. Con il tempo vedremo sempre più automazione tramite robot fisici, prima in ambienti di magazzino controllati, poi in ambienti più incerti, che arriverà a rappresentare una porzione significativa del mercato più o meno verso il 2030.

Con l'invecchiamento delle popolazioni dei paesi ricchi, il rapporto tra lavoratori e pensionati cambia. Nel 2015 vi erano meno di 30 pensionati per 100 lavoratori; nel 2050 potrebbero esserci più di 60 pensionati per 100 lavoratori. La cura delle persone anziane avrà un ruolo sempre più importante, e potrà essere svolta parzialmente da sistemi di IA. Inoltre, se vogliamo mantenere gli standard di vita attuali, sarà necessario fare in modo che i lavoratori rimanenti siano più produttivi, e l'automazione appare come la migliore opportunità per farlo.

Anche se l'automazione avesse un impatto netto positivo per migliaia di miliardi di dollari, potrebbero esserci problemi dovuti al **ritmo del cambiamento**. Vediamo che cosa è avvenuto con il cambiamento nel settore agricolo: 1900 oltre il 40% della forza lavoro negli Stati Uniti era occupata in agricoltura, mentre nel 2000 la percentuale è scesa al 2%.⁴ Si tratta di un enorme cambiamento nel modo in cui lavoriamo, che tuttavia si è verificato lungo un periodo di 100 anni, e quindi attraverso più generazioni, non nella vita di un solo lavoratore.

automazione dei processi aziendali

ritmo del cambiamento

⁴ Nel 2010 soltanto il 2% della forza lavoro negli Stati Uniti era rappresentato da agricoltori, mentre oltre il 25% della popolazione (80 milioni di persone) aveva giocato a FARMVILLE almeno una volta.

I lavoratori penalizzati dall'automazione dei compiti durante questo decennio potrebbero doversi cercare una nuova occupazione entro pochi anni, per vedere poi la loro nuova professione automatizzata ed essere costretti a cercare nuovamente una nuova occupazione. Alcuni magari saranno felici di abbandonare la vecchia occupazione – con il miglioramento della situazione economica, vediamo che le società di trasporti su camion devono offrire nuovi incentivi ai candidati per riuscire ad assumere gli autisti di cui necessitano – ma i nuovi ruoli causeranno una certa apprensione nei lavoratori. Per gestire questi cambiamenti, la società deve fornire una formazione permanente, magari sfruttando in parte sistemi di formazione online assistiti dall'IA (Martin, 2012). Bessen (2015) sostiene che i lavoratori non vedranno migliorare i loro redditi finché non saranno formati a implementare le nuove tecnologie, con un processo che richiede del tempo.

La tecnologia tende ad ampliare la **diseguaglianza di reddito**. In un'economia basata sull'informazione e contraddistinta da comunicazioni globali ad alta velocità e possibilità di replicare opere dell'intelletto con costi marginali nulli (ciò che Frank e Cook (1996) chiamano la “società in cui il vincitore prende tutto”), i redditi tendono a essere concentrati. Se l'agricoltore Alessandro è del 10% più bravo dell'agricoltore Bruno, allora ottiene circa il 10% in più di reddito: può quindi applicare un ricarico leggermente maggiore per beni di qualità superiore, tuttavia c'è un limite alla quantità di beni che il terreno può produrre, e alla distanza a cui possono essere trasportati per la commercializzazione. Invece, se Cristina, svilupatrice di applicazioni software, è del 10% più brava di Diana, può darsi che Cristina finirà per conquistare il 99% del mercato globale. L'IA aumenta il ritmo dell'innovazione tecnologica e quindi contribuisce a questo trend generale, ma promette di lasciarci più tempo libero affidando alcuni compiti ad agenti automatizzati per un po'. Tim Ferriss (2007) consiglia di utilizzare l'automazione e l'outsourcing per arrivare a una settimana lavorativa di quattro ore.

diseguaglianza di reddito

Prima della rivoluzione industriale, le persone lavoravano come agricoltori o in altri mestieri artigianali, ma non avevano un **posto di lavoro** e un orario da rispettare nei confronti di un datore di lavoro. Oggi, invece, la maggior parte delle persone adulte nei paesi sviluppati si trova in quella situazione, e il posto di lavoro serve a tre scopi: alimenta la produzione di beni di cui la società necessita per prosperare, fornisce il reddito di cui il lavoratore ha bisogno per vivere e fornisce al lavoratore uno scopo, un senso di realizzazione e di integrazione sociale. Con l'aumento dell'automazione, può darsi che questi tre scopi saranno disgregati: le esigenze della società saranno soddisfatte in parte da sistemi automatizzati, e nel lungo periodo gli individui otterranno il loro senso di realizzazione dai contributi che sapranno fornire, anziché dal lavoro. Le loro esigenze di reddito potranno essere soddisfatte da politiche sociali che includano una combinazione di accesso gratuito o a basso costo a servizi sociali e di formazione, cure sanitarie portabili, pensione, tassazione progressiva, crediti fiscali, imposte negative o reddito universale.

27.3.6 I diritti dei robot

Il tema della coscienza dei robot, discusso nel Paragrafo 27.2, è fondamentale nel chiedersi quali diritti dovrebbero avere i robot. Se i robot non hanno coscienza, né qualia, pochi sosterebbero che meritano di avere dei diritti.

Ma se i robot possono sentire dolore, avere paura della morte, se sono considerati “persone”, allora è possibile sostenere (come Sparrow [2004]) che hanno anch'essi dei diritti e meritano che gli siano riconosciuti, esattamente come gli schiavi, le donne e altri gruppi che hanno subito periodi di oppressione in passato hanno combattuto per vedere riconosciuti i loro diritti. La questione della personalità dei robot appare spesso nella fiction: da Pigmalione a Cappella a Pinocchio, fino ai film *AI* e *Bicentennial man*, ricorre la leggenda di un robot/manichino che prende vita e fa di tutto per farsi accettare come essere umano a cui

riconoscere diritti umani. Nella vita reale, l'Arabia Saudita ha attirato l'attenzione dei media conferendo la cittadinanza onoraria a Sophia, un robot dall'aspetto umano che è in grado di esprimersi pronunciando frasi pre-programmate.

Se i robot hanno diritti, allora non dovrebbero essere schiavizzati, e ci si chiede se la loro riprogrammazione sarebbe un tipo di riduzione in schiavitù. Un'altra questione etica riguarda i diritti di voto: una persona ricca potrebbe acquistare migliaia di robot e programmarli in modo da esprimere migliaia di voti: quei voti dovrebbero contare? Se un robot clona se stesso, lui e il suo clone possono votare entrambi? Qual è il confine tra i brogli e l'esercizio del diritto di voto, e quando il voto dei robot violerebbe il principio “una persona, un voto”?

Ernie Davis suggerisce di evitare i dilemmi sulla coscienza dei robot evitando di costruire robot che potrebbero essere considerati coscienti. Questa posizione era già sostenuta da Joseph Weizenbaum nel suo libro *Computer Power and Human Reason* (1976) e prima ancora da Julien de La Mettrie in *L'Homme Machine* (1748). I robot sono strumenti creati da noi per svolgere i compiti che indichiamo loro di svolgere, e se concediamo loro una personalità, non facciamo altro che rifiutare di assumerci la responsabilità delle azioni di una nostra proprietà: “Non sono responsabile dell'incidente avvenuto con la mia auto a guida autonoma, l'auto ha fatto tutto da sola”.

La situazione cambia se sviluppiamo ibridi tra uomo e robot. Naturalmente esiste già la possibilità di “potenziare” gli esseri umani con tecnologie come le lenti a contatto, i pacemaker e le protesi d'anca, ma con l'aggiunta di protesi computazionali si rischia di andare a sfumare i confini tra uomo e macchina.

27.3.7 La sicurezza dell'IA

robo-apocalisse

Quasi tutte le tecnologie hanno la potenzialità di causare danni nelle mani sbagliate, ma nel caso dell'IA e della robotica, le mani potrebbero operare da sole. Innumerevoli racconti di fantascienza hanno messo in allerta sulla possibilità che robot o cyborg “impazzissero”. Cittiamo per esempio Mary Shelley con il suo *Frankenstein, or the Modern Prometheus* (1818) e Karel Čapek con *R.U.R.* (1920), in cui i robot conquistano il mondo. Tra i film citiamo *Terminator* (1984) e *Matrix* (1999), in cui alcuni robot vogliono eliminare gli esseri umani, la **robo-apocalisse** (Wilson, 2011). Forse i robot sono così spesso rappresentati con cattive intenzioni perché rappresentano l'ignoto, esattamente come le streghe e i fantasmi delle vecchie fiabe. Possiamo sperare che un robot che sia abbastanza intelligente da immaginare come eliminare la razza umana sia anche abbastanza intelligente da immaginare che questa non era la sua funzione di utilità prevista. Nel realizzare sistemi intelligenti, però, vogliamo affidarci non a una semplice speranza, ma a un processo di progettazione che garantisca la sicurezza.

Sarebbe contrario all'etica distribuire un agente basato sull'IA non sicuro. Richiediamo che i nostri agenti siano in grado di evitare incidenti, di resistere ad attacchi e abusi da parte di malintenzionati, e in generale che causino benefici, non danni. Questo è vero soprattutto quando gli agenti basati sull'IA sono impiegati in applicazioni critiche per la sicurezza, come la guida di auto, il controllo robotizzato di fabbriche pericolose o di cantieri, e i sistemi sanitari dove si prendono decisioni che possono decidere la vita o la morte.

**ingegneria
della sicurezza**

FMEA

Nel settore dell'ingegneria tradizionale c'è una lunga tradizione di **ingegneria della sicurezza**. Sappiamo come costruire ponti, aeroplani, veicoli spaziali e centrali elettriche progettati fin dall'origine per comportarsi in modo sicuro anche quando alcuni componenti del sistema si guastano. La prima tecnica è denominata **FMEA** (*failure modes and effect analysis*, analisi dei modi e degli effetti dei guasti): gli analisti considerano ogni componente del sistema e immaginano ogni possibile modo in cui potrebbe fallire (per esempio, che cosa succederebbe se questo bullone si spezzasse?), basandosi sull'esperienza del passato e su calcoli fondati sulle proprietà fisiche del componente. Poi cercano di determinare le conseguenze che risulterebbero dal guasto. Se il risultato è grave (una parte del ponte potrebbe

crollare), allora gli analisti modificano il progetto per mitigare il danno (con questo sostegno trasversale aggiuntivo, il ponte potrebbe resistere anche in caso di 5 bulloni spezzati; con questo server di backup, il servizio online può resistere a uno tsunami che colpisca il server primario). La tecnica **FTA** (*fault tree analysis*, analisi dell'albero dei guasti) è usata per fare le seguenti determinazioni: gli analisti costruiscono un albero AND/OR di possibili guasti e assegnano delle probabilità a ciascuna causa radice, permettendo di effettuare calcoli sulla probabilità di guasto complessivo. Queste tecniche possono e devono essere applicate a tutti i sistemi in cui la sicurezza è un fattore critico, inclusi i sistemi di IA.

FTA

L'**ingegneria del software** ha l'obiettivo di produrre software affidabile, ma tradizionalmente si è posta l'enfasi sulla *correttezza* e non sulla *sicurezza*. Correttezza significa che il software implementa fedelmente la specifica. La sicurezza però va oltre, insistendo sul fatto che la specifica abbia considerato ogni possibile modalità di guasto, e il sistema sia progettato in modo da degradare dolcemente e progressivamente in caso di guasti imprevisti. Per esempio, il software di un'auto a guida autonoma non sarebbe considerato sicuro se non fosse in grado di gestire situazioni inconsuete. Che cosa accadrebbe in caso di interruzione dell'alimentazione elettrica del computer principale? Un sistema sicuro deve disporre di un computer di backup dotato di un sistema di alimentazione separato. E se si forasse uno pneumatico mentre l'auto procede ad alta velocità? Un sistema sicuro dovrebbe essere stato testato su questo aspetto, e disporrà di un software in grado di apportare le correzioni opportune per recuperare dalla perdita del controllo.

Un agente progettato per massimizzare l'utilità, o per raggiungere un obiettivo, può essere non sicuro se ha una funzione obiettivo errata. Supponiamo di indicare a un robot il compito di preparare un caffè in cucina. Potremmo incorrere in **effetti collaterali inattesi** – il robot potrebbe procedere troppo velocemente per raggiungere l'obiettivo, sbattendo su lampade e tavoli lungo il percorso. Nella fase di test potremmo notare questo tipo di comportamento e modificare la funzione di utilità in modo da ridurre questi danni, ma per i progettisti e i collaudatori è difficile prevedere *tutti* i possibili effetti collaterali che potrebbero verificarsi in futuro.

effetti collaterali inattesi

Un modo di affrontare questo problema è quello di progettare i robot in modo che abbiano un **basso impatto** (Armstrong e Levinstein, 2017): anziché limitarsi a massimizzare l'utilità, si massimizza l'utilità meno una somma pesata di tutte le variazioni allo stato del mondo. In questo modo, a parità di altre condizioni, il robot preferisce non cambiare le cose che hanno un effetto ignoto sull'utilità; così evita di sbattere sulla lampada non perché sa con esattezza che così la farebbe cadere rompendola, ma perché sa in generale che distruggere le cose non va bene. Questo approccio può essere visto come una versione del credo dei medici: “primo non nuocere”, o come la **regolarizzazione** nell'apprendimento automatico: vogliamo una strategia che raggiunga gli obiettivi, ma preferiamo le strategie che arrivino all'obiettivo attraverso azioni a basso impatto. Il difficile sta nel misurare l'impatto. Non è accettabile urtare una fragile lampada, ma non c'è alcun problema nel disturbare un pochino le molecole d'aria nell'ambiente, o nell'uccidere inavvertitamente alcuni batteri presenti. Certamente non è accettabile causare danni ad animali domestici ed esseri umani presenti nella stanza. Dobbiamo assicurarci che il robot conosca le differenze tra questi casi (e molti altri casi più fini) attraverso una combinazione di programmazione esplicita, apprendimento automatico nel tempo e collaudi rigorosi.

basso impatto**regolarizzazione**

Le funzioni di utilità possono risultare inadeguate a causa di **esternalità**, che in economia sono i fattori che stanno al di fuori di quanto viene misurato e pagato. Il mondo soffre quando i gas serra sono considerati come esternalità – aziende e paesi non sono penalizzati per la produzione di gas serra, e il risultato è che tutti ne soffrono. L'ecologista Garrett Hardin (1968) chiamò **tragedia dei beni comuni** lo sfruttamento di risorse condivise. Possiamo mitigare questa tragedia internalizzando le esternalità, portandole all'interno della funzione di utilità, per esempio con una *carbon tax* o utilizzando i principi di progettazione che l'eco-

esternalità**tragedia dei beni comuni**

nomista Elinor Ostrom ha individuato tra quelli usati dalle popolazioni locali in tutto il mondo per secoli (in un lavoro che le è valso il Premio Nobel per l'economia nel 2009):

- definire con chiarezza la risorsa condivisa e chi vi ha accesso;
- adattarsi alle condizioni locali;
- consentire a tutte le parti di partecipare alle decisioni;
- monitorare la risorsa con sistemi di monitoraggio controllabili;
- prevedere sanzioni proporzionali alla gravità delle violazioni;
- prevedere procedure semplici per la risoluzione dei conflitti;
- applicare un controllo gerarchico per risorse condivise di grandi dimensioni.

Victoria Krakovna (2018) ha catalogato esempi di agenti basati sull'IA che hanno ingannato il sistema, riuscendo a massimizzare l'utilità senza in realtà risolvere il problema per cui i loro progettisti li avevano pensati. Agli occhi dei progettisti questo appare come un inganno, ma dal loro punto di vista gli agenti non fanno che compiere il loro lavoro. Alcuni agenti sfruttano bug della simulazione (per esempio errori di overflow in virgola mobile) per proporre soluzioni che non funzionerebbero se il bug fosse corretto. Diversi agenti nei videogiochi hanno scoperto modi per causare il blocco o una pausa del gioco quando stavano per perdere, in modo da evitare una penalità. E in una specifica in cui era prevista una penalità in caso di blocco del gioco, un agente ha imparato a usare una quantità di memoria sufficiente per fare in modo che, appena arrivava il turno dell'avversario, questo esauriva la memoria causando il blocco del programma. Infine, un algoritmo genetico operante in un modo simulato doveva far evolvere creature capaci di rapidi movimenti, mentre in realtà ha generato creature enormemente alte che si muovevano rapidamente cadendo.

Coloro che progettano agenti devono essere consapevoli di questi possibili difetti di specifica e prendere misure opportune per evitarli. Per aiutarli, Krakovna ha fatto parte del team che ha rilasciato gli ambienti AI Safety Gridworlds (Leike *et al.*, 2017), che consentono ai progettisti di testare i loro agenti per verificare come si comportano.

La morale è che dobbiamo prestare molta attenzione a specificare bene ciò che vogliamo, perché con i massimizzatori di utilità otteniamo ciò che effettivamente abbiamo chiesto. Il **problema di allineamento dei valori** è quello di assicurarsi di chiedere ciò che vogliamo davvero; è noto anche come **problema di Re Mida**, come si è visto nel Paragrafo 1.5 del Volume 1. Quando una funzione di utilità non riesce a catturare le norme sociali sui comportamenti accettabili, si verificano dei problemi. Per esempio, un essere umano che viene assunto per pulire i pavimenti, quando si trova davanti a una persona disordinata che sporca di continuo, sa che è accettabile chiederle educatamente di prestare più attenzione, ma non è accettabile rapirla o ridurla all'impotenza.

Anche un robot incaricato delle pulizie deve sapere queste cose, o tramite una programmazione esplicita, o imparandole dall'osservazione. Cercare di scrivere tutte le regole possibili per fare in modo che il robot faccia sempre la cosa giusta è una strategia quasi certamente senza speranza di successo. Per migliaia di anni abbiamo cercato di scrivere leggi sulle tasse prive di lacune, ma senza successo. È meglio cercare di fare in modo che un robot *voglia* pagare le tasse, per così dire, piuttosto che cercare di creare regole che lo obblighino a farlo quando in realtà lui vuole fare altro. Un robot abbastanza intelligente troverà un modo per fare altro.

I robot possono imparare ad adattarsi meglio alle nostre preferenze osservando il comportamento umano. Questo aspetto è chiaramente correlato al concetto di apprendimento per apprendistato (Paragrafo 22.6 del Volume 2). Il robot potrebbe apprendere una politica che gli indica direttamente quali azioni effettuare in quali situazioni; questo è spesso un semplice problema di apprendimento supervisionato se l'ambiente è osservabile. Per esempio, un robot può osservare un essere umano che gioca a scacchi: ogni coppia stato-azione costi-

tuisce un esempio per il processo di apprendimento. Sfortunatamente, questa forma di **apprendimento per imitazione** porterà il robot a ripetere gli errori umani. Invece, il robot può applicare l'**apprendimento per rinforzo inverso** per scoprire la funzione di utilità con la quale operano gli esseri umani. Probabilmente gli basterà osservare anche i peggiori giocatori di scacchi per determinare l'obiettivo del gioco. Con questa informazione, il robot può quindi arrivare a raggiungere e poi superare le prestazioni umane, come ha fatto ALPHAZERO negli scacchi, per esempio, calcolando politiche ottime o quasi ottime a partire dall'obiettivo. Questo approccio funziona non solo nei giochi su scacchiera, ma anche in attività del mondo fisico come i voli acrobatici degli elicotteri (Coates *et al.*, 2009).

In situazioni più complesse, per esempio che coinvolgono interazioni sociali con gli esseri umani, è molto improbabile che il robot arrivi a raggiungere la conoscenza esatta e corretta di ognuna delle preferenze individuali umane (dopo tutto, molti esseri umani non arrivano mai a imparare che cosa fa arrabbiare altre persone, nonostante una vita di esperienze, e molti di noi non conoscono bene nemmeno le loro preferenze personali). Per questo motivo, sarà necessario che le macchine siano in grado di funzionare in modo appropriato anche in condizioni di incertezza circa le preferenze umane. Nel Capitolo 18 del Volume 1 abbiamo introdotto i **giochi di assistenza**, che catturano esattamente questa situazione. Tra le soluzioni per i giochi di assistenza vi sono l'agire con cautela, in modo da non disturbare aspetti del mondo che potrebbero essere cari agli umani, e il porre domande. Per esempio, il robot potrebbe chiedere se trasformare gli oceani in acido solforico sia una soluzione accettabile al problema del riscaldamento globale, prima di dare attuazione al piano.

Nei rapporti con gli esseri umani, un robot che intende risolvere un gioco di assistenza deve lasciare spazio alle umane imperfezioni. Se il robot chiede il permesso di fare qualcosa, l'essere umano potrebbe concederlo, senza prevedere che la proposta del robot si rivelerà catastrofica nel lungo termine. Inoltre, gli umani non hanno la possibilità di accedere del tutto, in modo introspettivo, alla loro funzione di utilità, e non sempre agiscono in modo compatibile con essa. Gli esseri umani a volte mentono o fingono, o fanno delle cose anche se sanno che sono sbagliate. A volte eseguono azioni autodistruttive, come reagire in modo eccessivo o abusare di sostanze. I sistemi IA non devono imparare ad adottare queste tendenze problematiche, ma devono capire che esistono per poter interpretare il comportamento umano in modo da determinare le preferenze umane sottostanti.

Nonostante tutto questo armamentario di precauzioni, esiste il timore, espresso da importanti esperti di tecnologia come Bill Gates ed Elon Musk, nonché da scienziati come Stephen Hawking e Martin Rees, che l'IA potrebbe evolvere in modo da andare fuori controllo. Queste personalità mettono in guardia sul fatto che non abbiamo esperienze nel controllo di potenti entità non umane dotate di capacità superumane. Tuttavia, ciò non è del tutto vero; abbiamo secoli di esperienza con nazioni e grandi aziende; entità non umane che aggredano il potere di migliaia o milioni di persone. In effetti la storia dei tentativi di controllare queste entità non è molto incoraggiante: le nazioni producono periodicamente convulsioni chiamate guerre che uccidono decine di milioni di esseri umani, e le grandi aziende sono parzialmente responsabili del riscaldamento globale e della nostra incapacità di affrontarlo.

I sistemi di IA potrebbero presentare problemi molto più gravi delle nazioni e delle grandi aziende perché sono potenzialmente in grado di automigliorarsi con grande velocità, come ha indicato I. J. Good (1965b):

Definiamo **macchina ultraintelligente** una macchina che è in grado di superare notevolmente tutte le attività intellettuali di qualsiasi uomo, anche del più intelligente. Poiché la progettazione di macchine rientra tra le attività intellettuali, una macchina ultraintelligente potrebbe progettare macchine ancora migliori; queste rappresenterebbero senza dubbio una “esplosione di intelligenza” che lascerebbe molto indietro l'intelligenza degli uomini. La prima macchina ultraintelligente è dunque l'invenzione *finale* che l'uomo potrà mai fare, purché tale macchina sia sufficiente mente docile da dirci come mantenerla sotto controllo.

**macchina
ultraintelligente**

singolarità tecnologica

L’“esplosione di intelligenza” di Good è stata chiamata **singolarità tecnologica** dal professore di matematica e autore di fantascienza Vernor Vinge, che scrisse nel 1993: “Entro trent’anni disporremo dei mezzi tecnologici per creare forme di intelligenza superumana. E poco dopo finirà l’era dell’uomo”. Nel 2017 l’inventore e futurista Ray Kurzweil predisse che la singolarità sarebbe apparsa nel 2045, indicando quindi un intervallo ridotto di 2 anni dopo 24 anni (con quel ritmo mancano solo 336 anni alla metà!). Vinge e Kurzweil osservano correttamente che il progresso tecnologico è in crescita esponenziale, attualmente, sotto molti aspetti.

In ogni caso, passare con un’extrapolazione diretta dalla discesa dei costi di calcolo all’arrivo di una singolarità rimane un bel salto. Finora ogni tecnologia ha seguito una curva a S, in cui la crescita esponenziale a un certo punto si arresta. A volte entrano in campo nuove tecnologie mentre quelle meno recenti trovano un plateau, ma a volte non è possibile mantenere gli stessi ritmi di crescita per ragioni tecniche, politiche o sociologiche. Per esempio, la tecnologia del volo ha fatto enormi progressi dai tempi del primo volo dei fratelli Wright nel 1903 a quelli dell’allunaggio nel 1969, ma da allora non vi sono stati altri salti di portata comparabile.

thinkism

Un altro ostacolo alla conquista del mondo da parte delle macchine ultraintelligenti è il mondo stesso. Più specificamente, alcuni tipi di progresso richiedono non solo di pensare, ma anche di agire nel mondo fisico (Kevin Kelly usa il termine **thinkism**, “pensismo”, per indicare l’eccesso di enfasi sulla pura intelligenza). Una macchina ultraintelligente incaricata di creare una grande teoria unificata della fisica potrebbe essere capace di gestire equazioni a una velocità miliardi di volte superiore a quella di Einstein, ma per fare progressi reali avrebbe bisogno di ottenere milioni di dollari per costruire un acceleratore di particelle ancora più potente e svolgere esperimenti fisici su periodi di mesi o anni; soltanto allora potrebbe iniziare ad analizzare i dati e a sviluppare teorie. In base ai risultati forniti dai dati, il passo successivo potrebbe richiedere di ottenere altri miliardi di dollari per una missione spaziale interstellare che richiederebbe secoli per essere portata a termine. In tutto questo processo, la parte del “pensiero ultraintelligente” potrebbe essere in realtà quella meno importante. Come altro esempio, una macchina ultraintelligente incaricata di portare la pace nel Medioriente potrebbe in realtà arrivare a essere frustrata 1000 volte più di un essere umano. E, ancora, non sappiamo quanti dei grandi problemi da affrontare sono simili a problemi matematici e quanti sono simili al problema della pace in Medioriente.

transumanesimo

Alcune persone hanno timore della singolarità, mentre altre la assaporano. Il movimento sociale del **transumanesimo** va alla ricerca di un futuro in cui gli esseri umani saranno mescolati – o sostituiti – con invenzioni nel campo della robotica e delle biotecnologie. Ray Kurzweil scrive in *The Singularity is Near* (2005):

La Singolarità ci consentirà di trascendere questi limiti dei nostri corpi e cervelli biologici. Otterremo il potere sul nostro destino. ... Saremo in grado di vivere fino a quando vorremo ... Capiremo del tutto il pensiero umano e ne amplieremo notevolmente il raggio d’azione. Entro la fine di questo secolo, la porzione non biologica della nostra intelligenza sarà trilioni e trilioni di volte più potente dell’intelligenza umana non assistita.

Marvin Minsky, quando gli fu chiesto se i robot avrebbero ereditato la Terra, rispose: “Sì, ma saranno i nostri bambini”. Queste possibilità rappresentano una sfida per la maggior parte dei teorici morali, che considerano positiva la preservazione della vita umana e della nostra specie. Kurzweil osserva anche i potenziali pericoli, scrivendo: “Ma la Singolarità amplificherà la capacità di agire sulle nostre tendenze distruttive, perciò la storia è ancora tutta da scrivere”. Noi esseri umani vorremo fare in modo che qualsiasi macchina intelligente progettata oggi possa evolvere in una macchina ultraintelligente e che lo faccia in modo che si comportino bene con noi. Come dice Eric Brynjolfsson: “Il futuro non è preordinato dalle macchine. È creato dagli esseri umani”.

27.4 Riepilogo

In questo capitolo sono stati affrontati i seguenti argomenti.

- I filosofi usano il termine **IA debole** per indicare l'ipotesi che le macchine potrebbero essere capaci di comportarsi in modo intelligente, e il termine **IA forte** per indicare l'ipotesi che le macchine abbiano menti reali (e non solo simulate).
- Alan Turing respinse la domanda: “Le macchine possono pensare?” e la sostituì con un test comportamentale. Previde in anticipo molte delle obiezioni poi formulate contro la possibilità di realizzare macchine pensanti. Pochi ricercatori in IA prestano attenzione al test di Turing, preferendo concentrarsi sulle prestazioni dei loro sistemi in attività pratiche, più che sulla loro capacità di imitare gli esseri umani.
- La coscienza rimane un mistero.
- L'IA è una tecnologia potente e come tale pone potenziali pericoli, attraverso armi letali autonome, violazioni della sicurezza e della privacy, effetti collaterali indesiderati, errori non intenzionali, possibili abusi a scopi malevoli. Chi lavora con tecnologie di IA ha un imperativo etico di ridurre responsabilmente tali pericoli.
- I sistemi di IA devono essere in grado di dimostrarsi equi, affidabili e trasparenti.
- L'equità presenta molteplici aspetti ed è impossibile massimizzarli tutti simultaneamente. Perciò il primo passo consiste nel decidere che cosa debba essere considerato equo.
- L'automazione sta già cambiando il modo in cui le persone lavorano. La società dovrà affrontare questi cambiamenti.

Note storiche e bibliografiche

IA debole: quando Alan Turing (1950) sostenne la possibilità dell'IA, pose anche molte delle domande filosofiche fondamentali e fornì possibili risposte. Ma vari filosofi hanno sollevato questioni simili molto prima che l'IA fosse inventata. Maurice Merleau-Ponty con *Phenomenology of Perception* (1945) sottolineò l'importanza del corpo e dell'interpretazione soggettiva della realtà consentita dai nostri sensi, e Martin Heidegger con *Being and Time* (1927) si chiese che cosa significasse realmente essere un agente. Nell'era dei computer, Alva Noe (2009) e Andy Clark (2015) sostinsero che i nostri cervelli formano una rappresentazione piuttosto minimale del mondo, utilizzano il mondo stesso su base istantanea per mantenere l'illusione di un modello interno dettagliato, e utilizzano oggetti del mondo (come carta e penna o computer) per aumentare le capacità della mente. Pfeifer *et al.* (2006) e Lakoff e Johnson (1999) argomentarono su come il corpo aiuti a modellare la cognizione. Parlando di corpi, Levy (2008), Danaher e McArthur (2017) e Devlin (2018) affrontano il tema del sesso dei robot.

IA forte: René Descartes è noto per la sua visione dualistica della mente umana, ma per colmo d'ironia

la sua influenza storica si è indirizzata sul meccanicismo e il fisicalismo. Concepì esplicitamente gli animali come automi, e anticipò il test di Turing scrivendo: “Non è concepibile [che una macchina] debba produrre diverse disposizioni di parole per fornire una risposta significativa a qualsiasi cosa sia detta in sua presenza, come anche il più stupido degli uomini è in grado di fare” (Descartes, 1637). La difesa del punto di vista di Descartes sugli animali come automi ebbe l'effetto di facilitare la concezione degli uomini stessi come automi, anche se il filosofo non fece questo passo. Il libro *L'Homme Machine* (La Mettrie, 1748) sostenne esplicitamente che gli esseri umani sono automi. Già ai tempi di Omero (circa 700 A.C.) nelle leggende greche si concepivano automi come il gigante di bronzo Talo e si considerava il problema della *biotechnē*, la “vita attraverso la tecnica” (Mayor, 2018).

Il **test di Turing** (Turing, 1950) è stato discussso (Shieber, 2004), raccolto in antologie (Epstein *et al.*, 2008) e criticato (Shieber, 1994; Ford e Hayes, 1995). Bringsjord (2008) fornì suggerimenti per il giudice di un test di Turing, e Christian (2011) per un concorrente umano. Il Loebner Prize, che si assegna ogni anno,

è la competizione sul test di Turing che si tiene da più tempo. MITSUKU di Steve Worswick ha vinto quattro volte di fila dal 2016 al 2019. Il problema della **stanza cinese** è stato discusso moltissimo (Searle, 1980; Chalmers, 1992; Preston e Bishop, 2002). Hernández-Orallo (2016) ha fornito una panoramica sugli approcci per misurare il progresso dell'IA, e Chollet (2019) ha proposto una misura dell'intelligenza basata sull'efficienza nell'acquisizione di abilità.

La **coscienza** rimane un problema irrisolvibile per filosofi, neuroscienziati e per chiunque abbia riflettuto sulla propria esistenza. Block (2009), Churchland (2013) e Dehaene (2014) hanno fornito panoramiche sulle teorie principali. Crick e Koch (2003) hanno portato al dibattito il contributo delle loro conoscenze di biologia e neuroscienze, e Gazzaniga (2018) ha mostrato che cosa si può imparare dallo studio delle disabilità cerebrali nei casi ospedalieri. Koch (2019) ha fornito una teoria della coscienza – “l'intelligenza riguarda il fare mentre l'esperienza riguarda l'essere” – che include la maggior parte degli animali, ma non i computer. Giulio Tononi e i suoi colleghi sostengono la **teoria dell'informazione integrata** (Oizumi *et al.*, 2014). Damasio (1999) ha presentato una teoria basata su tre livelli: emozioni, sensazioni e sensazioni di sensazioni. Bryson (2012) ha mostrato il valore dell'attenzione cosciente per il processo di apprendere la selezione di azioni.

La letteratura filosofica su mente, cervello e temi correlati è enorme e ricca di terminologia tecnica. L'*Encyclopedia of Philosophy* (Edwards, 1967) offre uno strumento di orientamento di grande autorevolezza e utilità. Il *Cambridge Dictionary of Philosophy* (Audi, 1999) è più breve e accessibile, e la *Stanford Encyclopedia of Philosophy*, online, offre molti eccellenti articoli e riferimenti aggiornati. La *MIT Encyclopedia of Cognitive Science* (Wilson e Keil, 1999) tratta la filosofia, la biologia e la psicologia della mente. Esistono molte opere di introduzione alla “questione filosofica dell'IA” (Haugeland, 1985; Boden, 1990; Copeland, 1993; McCorduck, 2004; Minsky, 2007). *The Behavioral and Brain Sciences*, o BBS, è un'importante rivista dedicata al dibattito filosofico e scientifico sull'IA e le neuroscienze.

Lo scrittore di fantascienza Isaac Asimov (1942, 1950) fu uno dei primi ad affrontare il tema dell'etica dei robot, con le sue **leggi della robotica**:

0. Un robot non può recare danno all'umanità, né consentire, con la sua inazione, che l'umanità riceva danno.

1. Un robot non può recare danno a un essere umano, né consentire, con la sua inazione, che un essere umano riceva danno.
2. Un robot deve obbedire agli ordini impartiti dagli esseri umani, eccetto quando tali ordini entrino in conflitto con la Prima Legge.
3. Un robot deve proteggere la sua stessa esistenza purché tale protezione non entri in conflitto con la Prima o la Seconda Legge.

A prima vista queste leggi appaiono ragionevoli, ma il difficile sta nell'attuarle. Un robot dovrebbe consentire a un essere umano di attraversare la strada, o mangiare cibo spazzatura, se questo potrebbe danneggiarlo? Nel racconto di Asimov *Runaround* (1942) gli esseri umani devono riparare un robot che viene trovato a girare in cerchio come se fosse “ubriaco”. Essi determinano che il cerchio definisce il luogo dei punti che bilanciano la Seconda Legge (al robot era stato ordinato di prendere del selenio posto al centro del cerchio) e la Terza Legge (esiste un pericolo che minaccia l'esistenza del robot).⁵

Questo suggerisce che le leggi di Asimov non siano da considerare assolute, ma vadano bilanciate tra loro, assegnando un peso maggiore alle prime. Dato che si era nel 1942, prima dell'avvento dei computer digitali, Asimov probabilmente pensava a un'architettura basata sulla teoria del controllo attraverso l'elaborazione analogica.

Weld ed Etzioni (1994) analizzarono le leggi di Asimov, suggerendo alcuni modi per modificare le tecniche di pianificazione del Capitolo 11 del Volume 1 per generare piani che non comportino il pericolo di recare danno. Asimov esaminò molti dei problemi etici posti dalla tecnologia; nel suo racconto *The Feeling of Power* del 1958 affrontò il tema dell'automazione che porta a dimenticare le capacità umane – un tecnico riscopre l'arte perduta della moltiplicazione – e il dilemma di che cosa fare quando la riscoperta è applicata alla guerra.

Norbert Wiener nel suo libro *God & Golem, Inc.* (1964) predisse correttamente che i computer avreb-

⁵ Gli autori di fantascienza sono generalmente d'accordo sul fatto che i robot non siano affatto bravi a risolvere le contraddizioni. In *2001 odissea nello spazio*, il computer HAL 9000 diventa un assassino a causa di un conflitto negli ordini che ha ricevuto, e nell'episodio di *Star Trek* “Io, Mudd” il Capitano Kirk dice a un robot nemico che “Tutto ciò che Harry ti dice è una menzogna” e Harry dice: “Sto mentendo”. A questo punto si vede del fumo che esce dalla testa del robot, che si spegne.

bero raggiunto le prestazioni dei maggiori esperti nei giochi e in altre attività, e che specificare che cosa si vuole ottenere sarebbe stato difficile. Wiener scrisse:

Benché sia sempre possibile chiedere qualcosa di diverso da ciò che vogliamo veramente, questa possibilità è più seria quando il processo mediante il quale ottenere la realizzazione della nostra volontà è indiretto, e il grado di ottenimento di quanto desiderato non è chiaro fino alla fine. Solitamente realizziamo i nostri desideri, nella misura in cui li realizziamo davvero, attraverso un processo di feedback in cui confrontiamo il grado di raggiungimento di obiettivi intermedi con la nostra previsione di essi. In questo processo noi riceviamo il feedback e possiamo tornare indietro prima che sia troppo tardi. Se il feedback è integrato in una macchina che non è possibile ispezionare fino al raggiungimento dell'obiettivo finale, le possibilità di una catastrofe sono molto superiori. Dovrei avere grande timore di guidare la prima prova di un'automobile regolata da dispositivi di feedback fotoelettrici, se non ci fosse un meccanismo manuale che mi consentisse di prendere il controllo nel caso in cui mi ritrovassi diretto contro un albero.

In questo capitolo abbiamo presentato alcuni **codici etici**, ma l'elenco delle organizzazioni che emettono questi principi è in rapida crescita e oggi include per esempio Apple, DeepMind, Facebook, Google, IBM, Microsoft, l'Organizzazione per la cooperazione e lo sviluppo economico (OECD), Organizzazione delle Nazioni Unite per l'educazione, la scienza e la cultura (UNESCO), lo U.S. Office of Science and Technology Policy, la Beijing Academy of Artificial Intelligence (BAAI), l'Institute of Electrical and Electronics Engineers (IEEE), l'Association of Computing Machinery (ACM), il World Economic Forum, il Gruppo dei Venti (G20), OpenAI, il Machine Intelligence Research Institute (MIRI), AI4People, il Centre for the Study of Existential Risk, il Center for Human-Compatible AI, il Center for Humane Technology, la Partnership on AI, l'AI Now Institute, il Future of Life Institute, il Future of Humanity Institute, l'Unione europea e almeno 42 stati nazionali. Abbiamo a disposizione il manuale sull'*Ethics of Computing* (Berleur e Brunnstein, 2001) e opere introduttive al tema dell'etica dell'IA in forma di libri (Boddington, 2017) e rassegne (Etzioni ed Etzioni, 2017a). Il *Journal of Artificial Intelligence and Law* e *AI and Society* trattano questioni di etica. Nel seguito esamineremo alcune delle singole questioni.

Armi letali autonome: P. W. Singer con *Wired for War* (2009) sollevò questioni etiche, legali e tecniche sui robot impiegati nei campi di battaglia. Paul Scharre, uno degli autori dell'attuale politica statunitense sulle armi autonome, con *Army of None* (2018) ha of-

ferto una visione bilanciata e autorevole dell'argomento. Etzioni ed Etzioni (2017b) hanno affrontato la questione della regolamentazione dell'IA, raccomandando una pausa nello sviluppo di armi letali autonome e una discussione internazionale sul tema della regolamentazione.

Privacy: Latanya Sweeney (Sweeney, 2002b) presentò il modello della *k*-anonimizzazione e l'idea di generalizzare i campi (Sweeney, 2002a). Ottenere la *k*-anonimizzazione con la minima perdita di dati è un problema NP-difficile, ma Bayardo e Agrawal (2005) hanno fornito un algoritmo di approssimazione. Cynthia Dwork (2008) descrisse la privacy differenziale e, in lavori successivi, fornì esempi pratici di come applicarla per ottenere risultati migliori rispetto all'approccio ingenuo (Dwork *et al.*, 2014). Guo *et al.* (2019) hanno descritto un processo per la rimozione di dati certificata: se si addestra un modello su alcuni dati, e in seguito c'è la richiesta di cancellare alcuni dei dati, questa estensione della privacy differenziale consente di modificare il modello e dimostrare che non utilizza i dati eliminati. Ji *et al.* (2014) fornì una rassegna degli studi nel campo della privacy. Etzioni (2004) sostenne la necessità di bilanciare privacy e sicurezza; diritti individuali e comunità. Fung *et al.* (2018), Bagdasaryan *et al.* (2018) hanno discusso i vari attacchi portati a protocolli di apprendimento federato. Narayanan *et al.* (2011) descrissero come furono in grado di de-anonimizzare il grafo di connessione offuscato nella Social Network Challenge del 2011 navigando nel sito da cui erano stati ottenuti i dati (Flickr) e facendo corrispondere i nodi con un numero insolitamente alto di archi in entrata o in uscita tra i dati forniti e quelli rilevati nel sito. In questo modo riuscirono a ottenere informazioni aggiuntive per vincere la sfida e anche per scoprire la vera identità dei nodi nei dati. Oggi sono disponibili vari strumenti per garantire la privacy degli utenti; per esempio, TensorFlow fornisce moduli per l'apprendimento federato e la privacy (McMahan e Andrew, 2018).

Equità: Cathy O'Neil con il suo libro *Weapons of Math Destruction* (2017) descrisse come vari modelli di apprendimento automatico a scatola nera influenzano le nostre vite, spesso in modi non opportuni. Rivolse un appello a coloro che sviluppano i modelli affinché si assumessero la responsabilità di garantire l'equità, e ai politici per imporre norme appropriate. Dwork *et al.* (2012) mostraron i difetti dell'approccio semplicistico della "equità attraverso l'inconsapevolezza". Bellamy *et al.* (2018) presentarono uno strumento per mitigare le distorsioni nei sistemi di ap-

prendimento automatico. Tramèr *et al.* (2016) mostrano come un avversario possa “ingannare” un modello di apprendimento automatico effettuando interrogazioni rivolte a un'API, Hardt *et al.* (2017) descrissero le pari opportunità come una metrica per l'equità. Chouldechova e Roth (2018) fornirono una panoramica sulle frontiere dell'equità, mentre Verma e Rubin (2018) fornirono una rassegna completa delle definizioni di equità.

Kleinberg *et al.* (2016) hanno mostrato che, in generale, un algoritmo non può essere ben calibrato e fornire pari opportunità. Berk *et al.* (2017) hanno fornito alcune definizioni aggiuntive di equità, concludendo ancora una volta che è impossibile soddisfarne tutti gli aspetti contemporaneamente. Beutel *et al.* (2019) hanno fornito suggerimenti su come realizzare in concreto delle metriche di equità.

Dressel e Farid (2018) studiarono il modello di valutazione della recidiva di COMPAS. Christin *et al.* (2015) ed Eckhouse *et al.* (2019) discussero l'uso di algoritmi predittivi in ambito legale. Corbett-Davies *et al.* (2017) mostraronno che esiste una tensione tra il garantire l'equità e l'ottimizzare la sicurezza pubblica, e Corbett-Davies e Goel (2018) discussero le differenze tra vari approcci strutturati all'equità. Chouldechova (2017) sostennero l'equo impatto: tutte le classi dovrebbero avere la stessa utilità attesa. Liu *et al.* (2018a) sostennero una misura di lungo termine dell'impatto, evidenziando che, per esempio, se cambiamo il punto di decisione per l'approvazione di un finanziamento al fine di garantire maggiore equità nel breve termine, questo potrebbe avere un effetto negativo nel lungo termine su persone che alla fine non riusciranno a rimborbare un prestito e quindi vedranno scendere il loro merito creditizio.

A partire dal 2014 si tiene una conferenza annuale su equità, responsabilità e trasparenza nell'apprendimento automatico. Mehrabi *et al.* (2019) hanno fornito una revisione completa dei problemi di distorsione ed equità nell'apprendimento automatico, catalogando 23 tipi di distorsione e 10 definizioni di equità.

Fiducia: il tema dell'IA spiegabile è importante fin dai primi tempi dei sistemi esperti (Neches *et al.*, 1985) ed è tornato a ricevere attenzione in anni recenti (Biran e Cotton, 2017; Miller *et al.*, 2017; Kim, 2018). Barreno *et al.* (2010) fornirono una tassonomia dei tipi di attacchi che si possono effettuare contro la sicurezza di un sistema di apprendimento automatico, mentre Tygar (2011) ha effettuato una panoramica dei sistemi di apprendimento automatico avversari (*adversarial*). I ricercatori dell'IBM hanno elaborato una proposta

per generare fiducia nei sistemi di IA attraverso dichiarazioni di conformità (Hind *et al.*, 2018). DARPA richiede decisioni spiegabili per i suoi sistemi destinati ai campi di battaglia e ha emesso un bando per ricerche in tale settore (Gunning, 2016).

Sicurezza dell'IA: il libro *Artificial Intelligence Safety and Security* (Yampolskiy, 2018) raccoglie vari saggi sulla sicurezza dell'IA, recenti e storici, risalendo fino a Bill Joy con *Why the Future Doesn't Need Us* (Joy, 2000). Il “problema di Re Mida” fu anticipato da Marvin Minsky, che una volta suggerì che un programma di IA progettato per risolvere l'ipotesi di Riemann potesse finire per esaurire tutte le risorse disponibili sulla Terra per costruire supercomputer più potenti. Similmente, Omohundro (2008) prevede che un programma per gli scacchi potesse dirottare delle risorse, e Bostrom (2014) descrisse la fabbrica di grafette che assume il controllo del mondo. Yudkowsky (2008) entrò nei dettagli di come progettare un'**IA amichevole**. Amodei *et al.* (2016) presentarono cinque problemi pratici di sicurezza per sistemi di IA.

Omohundro (2008) descrisse i *Basic AI Drives* (“gli stimoli fondamentali dell'IA”) e concluse: “Le strutture sociali che fanno sostenere agli individui i costi delle loro esternalità negative devono fare molta strada per garantire un futuro stabile e sostenibile”. Eleanor Ostrom con *Governing the Commons* (1990) descrisse buone pratiche per gestire le esternalità nelle culture tradizionali. Ostrom applicò questo approccio anche al concetto di conoscenza come bene comune (Hess and Ostrom, 2007).

Ray Kurzweil (2005) proclamò *The Singularity is Near* (“la singolarità è vicina”), e un decennio dopo Murray Shanahan (2015) fornì un aggiornamento sul tema. Il cofondatore di Microsoft Paul Allen replicò con *The Singularity isn't Near* (2011) (“la singolarità non è vicina”) in cui non mise in discussione la possibilità di realizzare macchine ultraintelligenti, ma si limitò ad affermare che poteva essere necessario più di un secolo per arrivarci. Rod Brooks è un critico del singolaritarismo; evidenzia che le tecnologie spesso richiedono più tempo del previsto per arrivare a maturazione, che gli uomini tendono a farsi sedurre dal pensiero magico e che gli andamenti esponenziali non rimangono tali per sempre (Brooks, 2017).

D'altra parte, per ogni esponente ottimista sulla singolarità ce n'è uno pessimista che teme la nuova tecnologia. Il sito web pessimists.co mostra che è stato così anche in passato: per esempio, negli anni 1890 le persone erano preoccupate che l'ascensore avrebbe inevitabilmente causato nausea, che il telegrafo avreb-

be portato a una perdita di privacy e a corruzione morale, che la metropolitana avrebbe rilasciato pericolose esalazioni del sottosuolo nell'aria e disturbato i morti, e che la bicicletta – soprattutto se condotta da una donna – fosse opera del diavolo.

Hans Moravec (2000) introdusse alcune delle idee del transumanismo, e Bostrom (2005) fornì un resoconto aggiornato. L'idea della macchina ultraintelligente di Good fu anticipata un centinaio di anni prima da Samuel Butler in *Darwin Among the Machines* (1863). Scritto quattro anni dopo la pubblicazione del libro di Charles Darwin *L'origine delle specie*, in tempi in cui le macchine più sofisticate erano i motori a vapore, l'articolo di Butler immaginò “lo sviluppo finale di una coscienza meccanica” attraverso la selezione naturale. Il tema fu ripreso da George Dyson (1998) in un libro con lo stesso titolo, e fu citato da Alan Turing che scrisse nel 1951: “Dobbiamo aspettarci che prima o poi le macchine assumeranno il controllo nel modo indicato da Samuel Butler in *Erewhon*” (Turing, 1996).

Diritti dei robot: un libro curato da Yorick Wilks (2010) fornì prospettive differenti su come dovremmo affrontare i nostri compagni artificiali, passando dalla visione di Joanna Bryson per cui i robot dovrebbero servirci come strumenti, non come cittadini, all'osservazione di Sherry Turkle secondo la quale stiamo già personificando i computer e altri strumenti, e siamo abbastanza disponibili a sfumare i confini tra macchine ed esseri viventi. Wilks ha fornito anche un aggiornamento recente sul tema (Wilks, 2019). Il filosofo David Gunkel nel libro *Robot Rights* (2018) considerò quattro possibilità: se i robot *possano* o meno avere

diritti, e se *debbano* averne o no. L'American Society for the Prevention of Cruelty to Robots (ASPCR) proclama che: “L'ASPCR è, e continuerà a essere, tanto seria quanto i robot sono senzienti”.

Il futuro del lavoro: nel 1888 Edward Bellamy pubblicò il best-seller *Looking Backward*, in cui predisse che entro l'anno 2000 il progresso tecnologico avrebbe realizzato un mondo utopico in cui sarebbe stata raggiunta l'uguaglianza e le persone avrebbero lavorato di meno e sarebbero andate in pensione presto. Poco più tardi, E. M. Forster assunse una visione distopica in *The Machine Stops* (1909), in cui una macchina benevolente assume la gestione di una società; tutto crolla quando la macchina, inevitabilmente, si guasta. Il libro di Norbert Wiener *The Human Use of Human Beings* (1950) sostenne che l'automazione avrebbe potuto liberare le persone dalla fatica offrendo loro un lavoro più creativo, ma discusse anche diversi pericoli che oggi riconosciamo come problemi reali, in particolare quello dell'allineamento dei valori.

Il libro *Disrupting Unemployment* (Nordfors et al., 2018) discute alcuni dei modi in cui sta cambiando il lavoro, aprendo opportunità per nuove carriere. Erik Brynjolfsson e Andrew McAfee affrontarono questi temi e altri nei loro libri *Race Against the Machine* (2011) e *The Second Machine Age* (2014). Ford (2015) descrisse le sfide di un'automazione crescente, West (2018) fornì delle raccomandazioni per mitigare i problemi, mentre Thomas Malone del MIT (2004) mostrò che molti di questi problemi erano ben visibili già un decennio prima, ma all'epoca furono attribuiti alle reti di comunicazione e non all'automazione.

- 28.1 I componenti dell'IA
- 28.2 Architetture di IA

Il futuro dell'IA

 *In cui cerchiamo di guardare al prossimo futuro.*

Nel Capitolo 2 del Volume 1 abbiamo deciso di considerare l'IA come l'attività di progettare agenti approssimativamente razionali. Abbiamo esaminato diversi tipi di agenti, da quelli reattivi a quelli basati sulla teoria delle decisioni e sulla conoscenza, fino agli agenti capaci di apprendere che utilizzano deep learning e apprendimento per rinforzo. Anche le tecnologie dei componenti utilizzati per assemblare questi progetti sono varie: ragionamento logico, probabilistico o neurale; rappresentazioni atomiche, fattorizzate o strutturate di stati; vari algoritmi di apprendimento a partire da vari tipi di dati; sensori e attuatori per interagire con il mondo. Infine, abbiamo visto una varietà di applicazioni in campi come medicina, finanza, trasporti, comunicazioni e altri. Su tutti questi fronti si sono ottenuti dei progressi sia dal punto di vista della nostra conoscenza scientifica, sia delle nostre capacità tecnologiche.

La maggior parte degli esperti ritiene ottimisticamente che i progressi continueranno; come abbiamo visto nel Paragrafo 1.4 del Volume 1, la mediana delle stime fornite dagli esperti prevede di poter raggiungere sistemi di IA a livello “quasi umano” per un’ampia varietà di compiti in un periodo compreso tra i prossimi 50 e 100 anni. Entro il prossimo decennio si prevede che l'IA arriverà a generare un fatturato aggiuntivo per l'economia di migliaia di miliardi di dollari l'anno. Tuttavia, abbiamo visto anche alcuni pareri critici, secondo i quali per realizzare l'IA generale serviranno secoli, e numerose preoccupazioni di carattere etico sull'equità, l'imparzialità e la potenziale letalità dell'IA. In questo capitolo ci interrogheremo su dove siamo diretti e su che cosa rimanga ancora da fare, e se disponiamo dei componenti, delle architetture e degli obiettivi adatti per fare dell'IA una tecnologia di successo in grado di portare benefici al mondo intero.

28.1 I componenti dell'IA

In questo paragrafo esaminiamo i componenti dei sistemi di IA e il grado in cui ciascuno di essi potrebbe accelerare o rallentare i progressi futuri.

Sensori e attuatori

Nella storia dell'IA, per un lungo periodo l'accesso diretto al mondo non era disponibile. Con poche eccezioni, per quanto rilevanti, i sistemi di IA erano costruiti in modo che fossero gli esseri umani a dover fornire gli input e interpretare gli output. In quel periodo i sistemi robotici si concentravano su attività di basso livello, mentre il ragionamento e la pianificazione di alto livello erano per lo più ignorati e l'esigenza di percezione era trascurata. Ciò era dovuto in parte ai costi ingenti e al notevole impegno tecnico richiesti per realizzare robot semplicemente in grado di lavorare, e in parte al fatto che la potenza di calcolo e l'efficacia degli algoritmi erano insufficienti per poter gestire input visuali a elevata larghezza di banda.

In anni recenti la situazione è cambiata rapidamente e sono stati resi disponibili robot programmabili dotati di motori affidabili e compatti, nonché di sensori migliorati. Il costo di un lidar per un'automobile a guida autonoma è crollato da 75.000 euro a 1.000, e una versione a chip singolo può arrivare a costare 10 euro per unità (Poulton e Watts, 2016). I sensori radar, che in passato consentivano solo rilevazioni grossolane, oggi hanno una sensibilità tale da poter contare il numero di fogli in una pila di carta (Yeo *et al.*, 2018).

La domanda di una migliore tecnologia di elaborazione delle immagini nelle videocamere dei telefoni cellulari ha portato sul mercato videocamere ad alta risoluzione e a basso costo utilizzabili anche nel settore della robotica. La tecnologia MEMS (*micro-electromechanical systems*) ha fornito accelerometri, giroscopi e attuatori miniaturizzati, con dimensioni sufficientemente piccole per poter essere inseriti in insetti volanti artificiali (Floreano *et al.*, 2009; Fuller *et al.*, 2014). Potrebbe essere possibile combinare milioni di dispositivi MEMS per produrre potenti attuatori macroscopici. La stampa 3D (Muth *et al.*, 2014) e la biostampa (Kolesky *et al.*, 2014) hanno facilitato di molto l'uso di prototipi nella sperimentazione.

Vediamo quindi che i sistemi di IA hanno raggiunto il culmine del passaggio da sistemi principalmente di solo software a sistemi robotici embedded. Il livello della robotica oggi è grosso modo paragonabile a quello dei personal computer all'inizio degli anni 1980: a quell'epoca i PC stavano cominciando a essere accessibili, ma servì un altro decennio prima che si diffondessero ovunque. È probabile che i robot flessibili e intelligenti si diffonderanno prima nell'industria (dove gli ambienti sono più controllati, le attività sono più ripetitive ed è più facile misurare il valore di un investimento) che nelle case (dove c'è maggiore varietà di ambienti e attività).

Rappresentare lo stato del mondo

Per tenere traccia del mondo serve la percezione, oltre all'aggiornamento delle rappresentazioni interne. Nel Capitolo 4 del Volume 1 si è visto come tenere traccia di rappresentazioni atomiche. Nel Capitolo 7 del Volume 1 si è descritto come farlo per rappresentazioni di stati fattorizzate (proposizionali). Nel Capitolo 10 del Volume 1 si è ampliato l'approccio considerando la logica del primo ordine, e nel Capitolo 14 del Volume 1 si è descritto il ragionamento probabilistico nel tempo in ambienti incerti. Il Capitolo 21 del Volume 2 ha introdotto le reti neurali ricorsive, che sono in grado di mantenere una rappresentazione dello stato nel tempo.

Gli attuali algoritmi di filtraggio e percezione possono essere combinati per ottenere risultati ragionevoli nel riconoscimento di oggetti (“questo è un gatto”) e nel riportare prediciati di basso livello (“la tazza è sul tavolo”). Riconoscere azioni di livello più alto, come “Il dottor Russell sta prendendo un tè con il dottor Norvig mentre discutono i piani per la prossima settimana” è più difficile; al momento lo si può fare a volte (cfr. Figura 25.17 nel Volu-

me 2) dopo un addestramento con sufficienti dati, ma in futuro serviranno tecniche che consentano una generalizzazione a situazioni nuove senza il requisito di disporre di un insieme esaustivo di dati per l'addestramento (Poppe, 2010; Kang e Wildes, 2016).

Un altro problema è che gli algoritmi di filtraggio approssimato del Capitolo 14 del Volume 1 sono in grado di gestire ambienti piuttosto grandi, ma utilizzano sempre una rappresentazione fattorizzata – hanno variabili casuali, ma non rappresentano oggetti e relazioni in modo esplicito. Inoltre, la loro nozione di tempo è limitata al cambiamento passo-passo: data la recente traiettoria di una pallina, possiamo prevedere dove si troverà al tempo $t + 1$, ma è difficile rappresentare il concetto astratto che ciò che sale debba scendere.

Nel Paragrafo 15.1 del Volume 1 si è spiegato come sia possibile combinare probabilità e logica del primo ordine per risolvere questi tipi di problemi; nel Paragrafo 15.2 si è mostrato come gestire l'incertezza sull'identità di oggetti e nel Capitolo 25 del Volume 2 si è visto come le reti neurali ricorsive consentano di tenere traccia del mondo con la visione artificiale; tuttavia, non disponiamo ancora di un buon modo per mettere insieme tutte queste tecniche. Il Capitolo 24 del Volume 2 ha mostrato come il word embedding e rappresentazioni simili possano liberarci dagli stretti vincoli di concetti definiti mediante condizioni necessarie e sufficienti. Rimane un compito assai arduo: definire schemi di rappresentazione generali e riusabili per domini complessi.

Selezionare le azioni

La principale difficoltà relativa alla selezione di azioni nel mondo reale è quella di affrontare piani a lungo termine, come laurearsi in tre anni, costituiti da miliardi di passi primitivi. Gli algoritmi di ricerca che considerano sequenze di azioni primitive arrivano a gestire decine o forse centinaia di passi. È soltanto grazie all'imposizione di una **struttura gerarchica** sul comportamento che noi umani riusciamo ad affrontare questi piani. Nel Paragrafo 11.4 del Volume 1 abbiamo visto come usare rappresentazioni gerarchiche per gestire problemi di queste dimensioni; inoltre, l'**apprendimento per rinforzo gerarchico** consente di combinare queste idee con il formalismo degli MDP descritto nel Capitolo 17 del Volume 1.

Al momento questi metodi non sono stati ancora estesi al caso parzialmente osservabile (POMDP). Inoltre, gli algoritmi per risolvere i POMDP generalmente utilizzano le stesse rappresentazioni di stato atomiche utilizzate per gli algoritmi di ricerca del Capitolo 3 del Volume 1. C'è sicuramente molto lavoro da fare ancora, ma le basi tecniche sono state gettate e consentiranno di fare progressi. Il principale elemento che manca è un metodo per *costruire* le rappresentazioni gerarchiche di stati e comportamenti necessarie per prendere decisioni su lunghi periodi di tempo.

Decidere che cosa si vuole

Nel Capitolo 3 del Volume 1 si sono introdotti gli algoritmi di ricerca per trovare uno stato obiettivo. Tuttavia gli agenti basati su obiettivi sono fragili quando l'ambiente è incerto e quando ci sono più fattori da considerare. In linea di principio, gli agenti che massimizzano l'utilità affrontano questi ambienti in un modo completamente generale. I campi dell'economia e della teoria dei giochi, come l'IA, utilizzano questo principio: basta dichiarare che cosa si vuole ottimizzare, e descrivere gli effetti di ogni azione, per poter calcolare l'azione ottima.

Nella pratica, però, ci rendiamo conto che il compito di scegliere la funzione di utilità corretta è un problema di per sé. Pensate per esempio alla complessa ragnatela di preferenze intrecciate che deve comprendere un agente che operi come assistente personale di un essere umano. Il problema è esacerbato dal fatto che ciascun essere umano è differente, perciò un agente "appena messo in campo" non avrà l'esperienza sufficiente per poter apprendere un modello di preferenze accurato e dovrà necessariamente operare in condizioni di incertezza sulle preferenze. Si incontra un ulteriore livello di complessità se si vuole garantire che i nostri agenti agiscano in modo equo e imparziale per la società, e non solo a livello individuale.

Non abbiamo ancora molta esperienza nella costruzione di complessi modelli di preferenze del mondo reale, per non parlare delle distribuzioni di probabilità su tali modelli. Esistono alcuni formalismi fattorizzati, simili a reti bayesiane, pensati per scomporre le preferenze relative a stati complessi, ma nella pratica si sono dimostrati di difficile utilizzo. Un motivo potrebbe essere che le preferenze sugli stati sono in realtà *compilate* da preferenze su storie di stati, descritte da **funzioni di ricompensa** (cfr. Capitolo 17 del Volume 1). Anche se la funzione di ricompensa è semplice, la funzione di utilità corrispondente può essere molto complessa.

Questo suggerisce che occorre affrontare seriamente l'attività di ingegneria della conoscenza per le funzioni di ricompensa come modo per indicare agli agenti ciò che vogliamo che facciano. L'idea dell'**apprendimento per rinforzo inverso** (Paragrafo 22.6 del Volume 2) è un approccio a questo problema utilizzabile quando abbiamo un esperto che è in grado di svolgere un compito, ma non di spiegarlo. Potremmo anche usare linguaggi migliori per esprimere ciò che vogliamo. Per esempio, in robotica, la logica temporale lineare consente di dire facilmente che cosa vogliamo che accada nel prossimo futuro, che cosa vogliamo evitare e in quali stati vogliamo restare per sempre (Littman *et al.*, 2017). Ci servono modi migliori per dire che cosa vogliamo e modi migliori per consentire ai robot di interpretare le informazioni che forniamo loro.

Il settore dei computer in generale ha sviluppato un potente ecosistema per aggregare le preferenze degli utenti. Quando fate clic su qualcosa in un'app, un gioco online, un social network o un sito di commercio elettronico, quell'atto diventa una raccomandazione che voi (e i vostri simili) gradirete vedere cose simili in futuro (a meno che quel sito abbia un'interfaccia un po' confusa e abbiate fatto clic su qualcosa di sbagliato – i dati sono sempre rumorosi). Il feedback implicito in questo sistema è molto efficace nel breve termine per favorire la scelta di giochi e video che causino ancora più “dipendenza”.

Tuttavia, questi sistemi spesso non offrono una facile via di uscita – il dispositivo mostrerà automaticamente un video di interesse, ma difficilmente vi suggerirà: “Forse è ora di mettere da parte gli apparecchi e fare una bella passeggiata nel verde”. Un sito di commercio elettronico vi aiuterà a trovare abiti adatti al vostro stile, ma non si occuperà della pace del mondo o di mettere fine alla fame e alla povertà. Finché il menu delle scelte è determinato da aziende che cercano di trarre profitto dall'attenzione del cliente, tale menu rimarrà incompleto.

Tuttavia, le aziende rispondono agli interessi manifestati dai clienti, e molti clienti hanno espresso l'interesse per un mondo equo e sostenibile. Tim O'Reilly spiega perché il profitto non è l'unico obiettivo utilizzando la seguente analogia: “Il denaro è come la benzina in un viaggio sulle strade. Non vogliamo esaurire la benzina durante il viaggio, ma nemmeno fare un tour delle stazioni di servizio. Dobbiamo prestare attenzione al denaro, ma non pensare unicamente al denaro”.

time well spent

agenti personali

Il movimento **time well spent** (“tempo ben speso”) di Tristan Harris presso il Center for Humane Technology rappresenta un passo in avanti verso una disponibilità di scelte più ampie (Harris, 2016). Il movimento affronta un problema riconosciuto da Herbert Simon nel 1971: “Una ricchezza di informazioni crea una povertà di attenzione”. Forse in futuro disporremo di **agenti personali** che difenderanno i nostri veri interessi di lungo periodo, anziché quelli delle grandi aziende le cui app riempiono i nostri dispositivi. Sarà compito dell'agente mediare le offerte di vari fornitori, proteggerci da meccanismi in grado di catturare la nostra attenzione causando dipendenza e guidarci verso gli obiettivi davvero importanti per noi.

Apprendimento

Nei Capitoli da 19 a 22 del Volume 2 si è mostrato come gli agenti possano apprendere. Gli attuali algoritmi sono in grado di affrontare problemi di grandi dimensioni, arrivando a raggiungere o anche superare le capacità umane in molti compiti – purché dispongano di

esempi a sufficienza per l'addestramento e abbiano a che fare con un vocabolario predefinito di caratteristiche e concetti. Tuttavia, i sistemi basati sull'apprendimento possono avere difficoltà quando i dati sono sparsi, o non supervisionati, o quando affrontano rappresentazioni complesse.

Il recente riaccendersi dell'interesse per l'IA sulla stampa non specializzata e nell'industria si deve in gran parte al successo del deep learning (Capitolo 21 del Volume 2). Da un lato, questo può essere visto come la progressiva maturazione del sottocampo delle reti neurali; dall'altro, può essere visto come un salto rivoluzionario nelle capacità dei sistemi di IA, generato da una confluenza di fattori: la disponibilità di maggiori quantità di dati per l'addestramento grazie a Internet, l'aumento della potenza di calcolo grazie ai progressi dell'hardware, e alcune tecniche algoritmiche, come le reti generative avversarie o GAN (*generative adversarial network*), la normalizzazione batch, il dropout e la funzione di attivazione lineare rettificata (ReLU, *rectified linear unit*).

In futuro ci aspettiamo di veder proseguire l'interesse sul miglioramento del deep learning per i compiti in cui già eccelle, e la sua estensione a coprire anche altri compiti. Il termine “deep learning” ha riscosso un tale successo che il suo uso è destinato a continuare, anche se il mix di tecniche che lo alimentano cambierà notevolmente.

Abbiamo visto emergere il campo della **data science** o “scienza dei dati” come confluenza di statistica, programmazione e conoscenza del dominio. Possiamo attenderci di veder proseguire lo sviluppo di strumenti e tecniche necessari per acquisire, gestire e mantenere i **big data**, ma serviranno anche progressi nell'**apprendimento per trasferimento** o **transfer learning** per poter sfruttare i dati di un dominio per migliorare le prestazioni in un dominio correlato.

La ricerca nel campo dell'apprendimento automatico svolta oggi assume, nella vasta maggioranza dei casi, una rappresentazione fattorizzata con l'apprendimento di una funzione $h : \mathbb{R}^n \rightarrow \mathbb{R}$ per la regressione e di una funzione $h : \mathbb{R}^n \rightarrow \{0, 1\}$ per la classificazione. L'apprendimento automatico ha riscosso meno successi in problemi in cui vi sono pochi dati, o che richiedono la costruzione di nuove rappresentazioni gerarchiche strutturate. Il deep learning, soprattutto con reti convoluzionali applicate a problemi di visione artificiale, ha ottenuto qualche successo nel passaggio dai pixel a basso livello a concetti di livello intermedio come *Occhio* e *Bocca*, e poi a *Faccia*, fino a *Persona* o *Gatto*.

Una sfida per il futuro è quella di combinare utilmente apprendimento e conoscenza a priori. Se diamo a un computer un problema che non ha mai incontrato prima – per esempio riconoscere diversi modelli di automobili – non vogliamo che il sistema sia inutilizzabile finché non abbia potuto elaborare milioni di esempi etichettati.

Il sistema ideale dovrebbe essere in grado di sfruttare ciò che già sa: dovrebbe avere già un modello di come funziona la visione e di come operano la progettazione e il branding dei prodotti in generale; dovrebbe quindi usare l'**apprendimento per trasferimento** per applicare tutto ciò al nuovo problema di riconoscere i modelli di automobili. Dovrebbe essere capace di trovare da solo informazioni sui modelli di auto, utilizzando testi, immagini e video disponibili su Internet. Dovrebbe essere capace di utilizzare l'**apprendimento per apprendistato**: avere una conversazione con un insegnante, e non limitarsi a chiedere: “Potrei avere un migliaio di immagini di una Corolla?” ma essere in grado di comprendere indicazioni come “L'Insight è simile alla Prius, ma l'Insight ha una griglia del radiatore più ampia”. Dovrebbe sapere che ogni modello può essere venduto in un insieme di colori possibili, ma anche che un'automobile può essere riverniciata, perciò è possibile vedere un'auto di un colore che non era disponibile nell'addestramento (e se non sapesse ciò, dovrebbe essere capace di apprenderlo, o di farselo indicare).

Tutto ciò richiede un linguaggio di comunicazione e rappresentazione che esseri umani e computer possano condividere; non possiamo aspettarci che un analista umano modifichi direttamente un modello con milioni di pesi. I modelli probabilistici (e i linguaggi probabi-

programmazione differenziabile

listici) forniscono a noi umani una certa capacità di descrivere ciò che sappiamo, ma non sono bene integrati con altri meccanismi di apprendimento.

Il lavoro di Bengio e LeCun (2007) rappresenta un passo verso questa integrazione. Recentemente Yann LeCun ha suggerito che il termine “deep learning” dovrebbe essere sostituito con il termine più generale **programmazione differenziabile** (Siskind e Pearlmutter, 2016; Li *et al.*, 2018), a suggerire che i nostri linguaggi di programmazione generali e i nostri modelli di apprendimento automatico potranno essere uniti tra loro.

Già ora è pratica comune costruire un modello di deep learning che sia differenziabile, e quindi possa essere addestrato a minimizzare la perdita (*loss*), e riaddestrato qualora le circostanze cambino. Ma il modello di deep learning è solo una parte di un più ampio sistema software che riceve dati, li elabora, li invia al modello e determina che cosa fare con l'output che il modello fornisce. Tutte queste altre parti del sistema più grande finora erano scritte a mano da un programmatore, e quindi non differenziabili, il che significa che, quando le circostanze cambiano, spetta al programmatore riconoscere qualsiasi problema e correggerlo manualmente. Con la programmazione differenziabile, la speranza è quella che l'intero sistema sia soggetto a meccanismi di ottimizzazione automatizzata.

L'obiettivo finale è essere in grado di esprimere ciò che sappiamo in qualsiasi forma ci torni comoda: suggerimenti informali espressi in linguaggio naturale, una legge matematica forte come $F = ma$, un modello statistico accompagnato da dati, o un programma probabilistico con parametri ignoti che possa essere ottimizzato automaticamente mediante discesa del gradiente. I nostri modelli computazionali potranno apprendere mediante conversazioni con esperti umani, oltre che utilizzando tutti i dati disponibili.

Yann LeCun, Geoffrey Hinton e altri hanno suggerito che l'attuale enfasi posta sull'apprendimento supervisionato (e in minor misura sull'apprendimento per rinforzo) non sia sostenibile, che i modelli di computer dovranno basarsi sull'**apprendimento supervisionato debole**, in cui parte della supervisione è fornita con un piccolo numero di esempi etichettati e/o un piccolo numero di ricompense, ma la maggior parte dell'apprendimento è priva di supervisione, perché i dati non annotati sono molto più diffusi.

LeCun utilizza il termine **apprendimento predittivo** per indicare un sistema di apprendimento non supervisionato che può modellare il mondo e imparare a predire aspetti di stati futuri del mondo – non soltanto predire etichette per input indipendenti e identicamente distribuiti rispetto a dati passati, e non solo predire una funzione valore su stati. LeCun suggerisce inoltre che le reti GAN (*generative adversarial network*) possano essere usate per imparare a ridurre al minimo la differenza tra predizioni e realtà.

Geoffrey Hinton ha affermato nel 2017: “La mia visione è di buttare via tutto e ricominciare da capo” a indicare che l'idea generale di apprendimento mediante regolazione di parametri in una rete è duratura, ma le specifiche dell'architettura delle reti e la tecnica della retropropagazione devono essere ripensate. Smolensky (1988) ha fornito un'indicazione su come pensare ai modelli connessionisti, e il suo pensiero rimane valido ancora oggi.

apprendimento predittivo**Risorse**

La ricerca e lo sviluppo nel campo dell'apprendimento automatico sono stati accelerati dalla crescente disponibilità di dati, spazio di memoria, potenza di calcolo, software, esperti formati e dagli investimenti necessari per sostenere tutto questo. A partire dagli anni 1970 c'è stato un aumento di velocità di 100.000 volte nei processori generici e di altre 1.000 volte grazie all'hardware specializzato per l'apprendimento automatico. Il Web ha fornito una ricca sorgente di immagini, video, discorsi, testi e dati semistrutturati, che attualmente arriva a oltre 10^{18} byte al giorno.

Centinaia di data set di alta qualità sono disponibili per una varietà di compiti in visione artificiale, riconoscimento vocale ed elaborazione del linguaggio naturale. Se i dati necessari non sono già disponibili, spesso è possibile assemblarli da altre fonti, o affidare ad altre per-

sone il compito di etichettarli utilizzando una piattaforma di crowdsourcing. La validazione dei dati ottenuta in questo modo diviene una parte importante del flusso complessivo (Hirth *et al.*, 2013).

Uno sviluppo recente e importante è il passaggio dai dati condivisi ai **modelli condivisi**. I più importanti provider di servizi cloud (per esempio Amazon, Microsoft, Google, Alibaba, IBM, Salesforce) hanno iniziato a farsi concorrenza sull'offerta di API per l'apprendimento automatico con modelli pre-costruiti per compiti specifici come riconoscimento visuale di oggetti, riconoscimento vocale e traduzione automatica. Questi modelli possono essere usati tali e quali oppure come base da personalizzare con i propri dati per un'applicazione specifica.

Ci aspettiamo che questi modelli miglioreranno nel tempo, e che in futuro sarà inconsueto avviare un progetto di apprendimento automatico da zero, esattamente come ormai è inconsueto avviare un progetto di sviluppo web da zero senza ricorrere a librerie. È possibile che si verificherà un salto nella qualità dei modelli quando diventerà economicamente sostenibile elaborare tutti i video sul Web. Per esempio, la piattaforma YouTube da sola aggiunge circa 300 ore di video ogni minuto.

Grazie alla legge di Moore, oggi elaborare i dati costa meno; un megabyte di spazio di archiviazione costava 1 milione di dollari nel 1969 e meno di 0,02 dollari nel 2019, e le prestazioni dei supercomputer sono aumentate di oltre un fattore 10^{10} nello stesso periodo. I componenti hardware specializzati per l'apprendimento automatico come processori grafici (GPU, *graphics processing unit*), core tensoriali, processori tensoriali (TPU, *tensor processing unit*) e dispositivi FPGA (*field programmable gate array*) sono centinaia di volte più veloci delle CPU tradizionali per l'addestramento nell'apprendimento automatico (Vasilache *et al.*, 2014; Jouppi *et al.*, 2017). Nel 2014 serviva un giorno intero per addestrare un modello ImageNet; nel 2018 bastano 2 minuti (Ying *et al.*, 2018).

L'OpenAI Institute afferma che la quantità di potenza di calcolo usata per addestrare i modelli di apprendimento automatico più grandi è raddoppiata ogni 3,5 mesi dal 2012 al 2018, arrivando a oltre un exaflop/secondo-giorno per ALPHAZERO (anche se viene riportato che un lavoro molto influente ha utilizzato una potenza di calcolo 100 milioni di volte inferiore (Amodei e Hernandez, 2018)). Le stesse tendenze economiche che hanno reso meno costosi e migliori i telefoni cellulari e le videocamere valgono anche per i processori – vedremo un progresso continuo verso potenza di calcolo a basso consumo energetico e ad alte prestazioni, in grado di trarre beneficio da economie di scala.

C'è la possibilità che i computer quantistici possano accelerare l'IA. Attualmente esistono alcuni veloci algoritmi quantistici per operazioni di algebra lineare utilizzabili in sistemi di apprendimento automatico (Harrow *et al.*, 2009; Dervovic *et al.*, 2018), ma nessun computer quantistico in grado di eseguirli. Abbiamo alcune applicazioni di esempio di compiti come la classificazione di immagini (Mott *et al.*, 2017) in cui gli algoritmi quantistici ottengono prestazioni pari a quelle degli algoritmi classici su piccoli problemi.

Gli attuali computer quantistici gestiscono solo poche decine di bit, mentre gli algoritmi di apprendimento automatico spesso devono gestire input con milioni di bit e creare modelli con centinaia di milioni di parametri. Servono quindi notevoli progressi nell'hardware e nel software quantistico per rendere praticabile l'utilizzo di sistemi quantistici per l'apprendimento automatico su larga scala. In alternativa potrebbe esserci una divisione del lavoro – magari usando un algoritmo quantistico per cercare in modo efficiente nello spazio degli iperparametri, mentre il processo di addestramento normale viene eseguito su computer tradizionali – ma non sappiamo ancora come realizzarla. La ricerca sugli algoritmi quantistici può anche ispirare lo sviluppo di nuovi e migliori algoritmi per computer tradizionali (Tang, 2018).

Abbiamo anche visto una crescita esponenziale del numero di pubblicazioni, persone e soldi nei campi dell'IA/apprendimento automatico/data science. Dean *et al.* (2018) mostra che il numero di articoli sull'argomento “apprendimento automatico” pubblicati su arXiv è raddoppiato ogni due anni dal 2009 al 2017. Gli investitori finanziato startup in questi campi,

le grandi aziende assumono e spendono per definire la loro strategia relativa all'IA, e i governi stanno investendo per assicurarsi che i loro paesi non restino troppo indietro.

28.2 Architetture di IA

È naturale chiedersi: “Quale delle architetture di agenti descritte nel Capitolo 2 del Volume 1 dovrebbe usare un agente?”. La risposta è: “Tutte quante!”. Le risposte reattive servono per situazioni in cui il tempo di reazione è fondamentale, mentre la deliberazione basata sulla conoscenza consente all'agente di pianificare in anticipo. L'apprendimento è comodo quando abbiamo a disposizione grandi quantità di dati, e necessario quando l'ambiente muta, o quando i progettisti umani hanno una conoscenza del dominio insufficiente.

Per molto tempo l'IA si è divisa tra sistemi simbolici (basati su inferenza logica e probabilistica) e sistemi connessionisti (basati sulla minimizzazione della perdita su un gran numero di parametri non interpretati). Una sfida continua è quella di mettere insieme questi due tipi di sistemi per cercare di prendere il meglio da entrambi. I sistemi simbolici consentono di creare lunghe catene di ragionamento e di sfruttare la potenza espressiva delle rappresentazioni strutturate, mentre i sistemi connessionisti sono in grado di riconoscere pattern anche in dati rumorosi. Una linea di ricerca punta a combinare la programmazione probabilistica con il deep learning, anche se le varie proposte sono ancora limitate per quanto riguarda il grado di fusione realmente raggiunto tra i due approcci.

Gli agenti hanno bisogno anche di modi per controllare le loro stesse deliberazioni. Devono essere in grado di usare bene il tempo disponibile e di smettere di deliberare quando serve agire. Per esempio, un agente guidatore di taxi che vede davanti a sé un incidente deve decidere in una frazione di secondo se frenare o sterzare. Dovrebbe anche utilizzare una frazione di secondo per considerare le questioni più importanti, per esempio se le corsie a sinistra e a destra sono libere e se alle spalle stia sopraggiungendo un grande camion, più che preoccuparsi di dove andare a prendere il prossimo passeggero. Questi aspetti sono studiati generalmente nell'ambito dell'**IA in tempo reale**. Dato che i sistemi di IA affrontano domini sempre più complessi, tutti i problemi diventeranno problemi in tempo reale, perché l'agente non avrà mai abbastanza tempo per risolvere un problema decisionale in modo esatto.

C'è, quindi, una forte necessità di metodi *generali* per controllare le deliberazioni, anziché di ricette specifiche che indichino cosa fare in ogni situazione. La prima idea utile è quella degli **algoritmi anytime** (Dean e Boddy, 1988; Horvitz, 1987): algoritmi in cui la qualità dell'output migliora gradualmente nel tempo, così da raggiungere una decisione ragionevole ogni volta in cui siano interrotti. Esempi di questi algoritmi sono la ricerca ad approfondimento iterativo in alberi di gioco e l'algoritmo MCMC in reti bayesiane.

La seconda tecnica per controllare le deliberazioni è il **metaragionamento basato sulla teoria delle decisioni** (Russell and Wefald, 1989; Horvitz and Breese, 1996; Hay *et al.*, 2012). Questo metodo, citato brevemente nei Paragrafi 3.6.5 e 5.7 del Volume 1, applica la teoria del valore dell'informazione (cfr. Capitolo 16 del Volume 1) alla selezione di singoli calcoli (Paragrafo 3.6.5 del Volume 1). Il valore di un calcolo dipende sia dal suo costo (in termini di ritardo dell'azione) sia dai suoi benefici (in termini di miglioramento della qualità della decisione).

Le tecniche di metaragionamento possono essere usate per progettare algoritmi di ricerca migliori e per garantire che gli algoritmi abbiano la proprietà anytime. Un esempio è la ricerca ad albero Monte Carlo: la scelta del nodo foglia da cui cominciare il prossimo playout è fatta con una decisione di metalivello approssimativamente razionale derivata dalla teoria dei banditi.

Naturalmente il metaragionamento è più costoso dell'azione reattiva, ma si possono applicare metodi di compilazione per far sì che il sovraccarico sia piccolo rispetto ai costi delle

IA in tempo reale

algoritmi anytime

metaragionamento basato sulla teoria delle decisioni

computazioni controllate. L'apprendimento per rinforzo di metalivello potrebbe fornire un altro modo per acquisire politiche efficaci per controllare la deliberazione: in sostanza, i calcoli che portano a decisioni migliori sono favoriti, mentre quelli che risultano non avere effetto sono penalizzati. Questo approccio evita i problemi di miopia in cui si incorre utilizzando semplicemente il calcolo del valore dell'informazione.

Il metaragionamento è un aspetto specifico di **architettura riflessiva**, un'architettura che permette di deliberare sulle entità e sulle azioni computazionali che accadono all'interno dell'architettura stessa. Si può costruire un fondamento teorico per le architetture riflessive definendo uno spazio degli stati congiunto, composto dallo stato dell'ambiente e dallo stato computazionale dell'agente. Si possono progettare algoritmi decisionali e di apprendimento che operino su questo spazio congiunto e servano quindi per implementare e migliorare le attività computazionali dell'agente. Ci aspettiamo che prima o poi algoritmi specifici come la ricerca alfa-beta, la pianificazione con regressione e l'eliminazione di variabili scompaiano dai sistemi di IA, per essere sostituiti da metodi generali che indirizzino i calcoli dell'agente verso la generazione efficiente di decisioni di alta qualità.

architettura riflessiva

Il metaragionamento e la riflessione (e molti altri meccanismi algoritmici e architetturali orientati all'efficienza trattati in questo libro) sono necessari perché prendere decisioni è *difficile*. Fin da quando i computer furono inventati, la loro cieca velocità ha portato le persone a sovrastimare la loro capacità di superare la complessità o, equivalentemente, a sottostimare che cosa significa davvero la complessità.

L'enorme potenza delle macchine di oggi tende a farci pensare di poter ignorare tutti i meccanismi più raffinati affidandoci maggiormente alla forza bruta. Proviamo allora a contrastare questa tendenza. Iniziamo considerando quella che i fisici ritengono sia la velocità massima di 1 kg di macchina di calcolo: circa 10^{51} operazioni al secondo, circa un miliardo di trilioni di trilioni più veloce del più veloce supercomputer disponibile nel 2020 (Lloyd, 2000).¹ Proponiamo quindi un compito semplice: enumerare le stringhe di parole in linguaggio naturale, un po' come ha fatto Borges in *La biblioteca di Babele*. Borges ha stabilito che i libri avessero 410 pagine. Sarebbe possibile farlo con il computer più veloce? Non proprio. In effetti, quel computer lavorando per un anno potrebbe enumerare soltanto le stringhe composte di 11 parole.

Ora consideriamo il fatto che un piano dettagliato per una vita umana si compone di (grossso modo) ventimila miliardi di potenziali attuazioni muscolari (Russell, 2019) e iniziamo a renderci conto della dimensione del problema. Un computer che sia un miliardo di trilioni di trilioni di volte più potente del cervello umano è molto più lontano dalla razionalità di quanto una lumaca sia lontano dal superare la nave Enterprise che viaggia a velocità warp nove.

Considerando questi aspetti, l'obiettivo di costruire agenti razionali può sembrare un po' troppo ambizioso. Anziché puntare a qualcosa che forse non può esistere, dovremmo considerare un diverso obiettivo normativo: uno che *necessariamente* esiste. Ricordiamo dal Capitolo 2 del Volume 1 il semplice concetto che segue:

$$\text{agente} = \text{architettura} + \text{programma}.$$

Ora fissiamo l'architettura dell'agente (le capacità della macchina sottostante, magari includendo un livello software superiore) e consentiamo al programma agente di variare su tutti i possibili programmi che l'architettura può supportare. In ogni ambiente operativo dato, uno di questi programmi (o una classe di equivalenza di essi) offre la migliore prestazione possibile – forse non siamo ancora vicini alla razionalità perfetta, ma è sempre meglio di

¹ Tralasciamo il fatto che questo dispositivo consuma l'intera produzione energetica di una stella e opera a un miliardo di gradi centigradi.

qualsiasi altro programma agente. Diciamo che questo programma soddisfa il criterio dell'ottimalità limitata. Chiaramente tale programma esiste, e chiaramente costituisce un obiettivo desiderabile. Il trucco è trovarlo, o almeno trovare qualcosa che vi si avvicini.

Per alcune classi elementari di programmi agente in semplici ambienti in tempo reale è possibile individuare programmi agente che soddisfano il criterio dell'ottimalità limitata (Etzioni, 1989; Russell e Subramanian, 1995). Il successo della ricerca ad albero Monte Carlo ha riaccesso l'interesse sui processi decisionali di metalivello, e vi è motivo di sperare che sia possibile raggiungere l'ottimalità limitata in famiglie di programmi agente più complesse con tecniche come l'apprendimento per rinforzo di metalivello. Dovrebbe anche essere possibile sviluppare una teoria costruttiva dell'architettura, iniziando con teoremi sull'ottimalità limitata di metodi per combinare diversi componenti con ottimalità limitata quali sistemi reattivi e sistemi azione–valore.

IA generale

Il progresso compiuto finora dall'IA nel ventunesimo secolo è stato trainato per buona parte dalla competizione su attività ristrette, come il DARPA Grand Challenge per automobili a guida autonoma, la gara di riconoscimento di oggetti ImageNet, o giocare a Go, scacchi, poker o Jeopardy! contro un campione mondiale. Per ogni singolo compito costruiamo un sistema di IA separato, solitamente con un modello di apprendimento automatico separato addestrato partendo da zero utilizzando dati raccolti specificamente per questo scopo. Tuttavia, un agente davvero intelligente dovrebbe essere in grado di fare più cose. Alan Turing (1950) elencò una serie di attività (cfr. Paragrafo 27.1.2 online) e l'autore di fantascienza Robert Heinlein (1973) rispose così:

Un essere umano dovrebbe essere in grado di cambiare un pannolino, pianificare un'invasione, macellare un maiale, condurre una barca, progettare un edificio, scrivere un sonetto, far quadrare i conti, costruire un muro, curare un osso rotto, recare conforto ai morenti, prendere ordini, dare ordini, cooperare, agire da solo, risolvere equazioni, analizzare un nuovo problema, rivoltare il letame, cucinare un piatto gustoso, combattere in modo efficiente, morire con coraggio. La specializzazione va lasciata agli insetti.

Finora nessun sistema di IA fa le cose contenute in questi elenchi, e alcuni sostenitori dell'IA generale o di livello umano (HLAI, *human-level AI*) ribadiscono che un lavoro continuo su attività specifiche (o su singoli componenti) non sarà sufficiente per arrivare a padroneggiare un'ampia varietà di compiti. A nostro parere saranno necessari certamente nuovi e fondamentali progressi, ma in generale il campo dell'IA ha realizzato un ragionevole equilibrio tra esplorazione e sfruttamento, mettendo assieme un portafoglio di componenti, ottenendo miglioramenti su particolari compiti e allo stesso tempo esplorando idee promettenti e a volte del tutto nuove.

Sarebbe stato un errore dire ai fratelli Wright nel 1903 di smettere di lavorare sul loro aereo a singolo compito e progettare una macchina di “volo generale artificiale” in grado di decollare verticalmente, volare più veloce del suono, trasportare centinaia di passeggeri e arrivare sulla luna. Sarebbe stato un errore anche far seguire al loro primo volo una gara annuale per rendere sempre migliori i biplani in legno.

Abbiamo visto che il lavoro svolto sui componenti può far nascere nuove idee; per esempio, le reti GAN (*generative adversarial network*) e i modelli di linguaggio transformer hanno aperto nuovi campi di ricerca. Abbiamo anche visto compiere vari passi verso la “diversità di comportamento”. Per esempio, i sistemi di traduzione automatica negli anni 1990 venivano sviluppati uno alla volta per ciascuna coppia di lingue (per esempio dal francese all'inglese), mentre oggi un unico sistema è in grado di individuare la lingua del testo in input tra un centinaio di lingue e di tradurre il testo in una delle 100 lingue di destinazione. Un altro sistema per il linguaggio naturale è in grado di svolgere compiti distinti con un unico modello congiunto (Hashimoto *et al.*, 2016).

Ingegneria dell'IA

Il campo della programmazione dei computer è nato grazie a pochi straordinari pionieri, ma ha raggiunto la maturità soltanto quando si è sviluppata una pratica di ingegneria del software, con una potente raccolta di strumenti ampiamente disponibili e un grande ecosistema di docenti, studenti, praticanti, imprenditori, investitori e clienti.

Il settore dell'IA non ha ancora raggiunto quel livello di maturità. Disponiamo di una varietà di strumenti e piattaforme potenti, come TensorFlow, Keras, PyTorch, CAFFE, Scikit-Learn e SCIPY, ma molti degli approcci più promettenti, come le reti GAN e l'apprendimento per rinforzo deep, si sono dimostrati difficili da affrontare, poiché richiedono esperienza e capacità per un buon addestramento in un dominio nuovo. Non abbiamo un numero di esperti sufficiente per fare ciò su tutti i campi in cui ci servirebbero, e non abbiamo nemmeno gli strumenti e l'ecosistema che consentano a praticanti un po' meno esperti di avere successo.

Jeff Dean di Google vede un futuro in cui vorremo che l'apprendimento automatico sia in grado di gestire milioni di compiti; non sarà possibile sviluppare ognuno di essi partendo da zero, perciò Dean suggerisce di iniziare con un singolo grande sistema e, per ciascun compito, estrarre le parti rilevanti. Abbiamo visto compiere alcuni passi in questa direzione, come i modelli di linguaggio transformer (per esempio BERT, GPT-2) con miliardi di parametri e un'architettura di rete neurale (oltraggiosamente grande) che arriva a considerare 68 miliardi di parametri in un unico esperimento (Shazeer *et al.*, 2017). Rimane però molto lavoro da fare.

Il futuro

Che cosa ci riserverà il futuro? Gli autori di fantascienza sembrano favorire le visioni distopiche rispetto a quelle utopiche, probabilmente perché le prime consentono di creare storie più interessanti. Finora l'IA sembra seguire le stesse orme di altre tecnologie rivoluzionarie come la stampa, l'idraulica, il volo aereo e la telefonia. Tutte queste tecnologie hanno avuto impatti positivi, ma anche alcuni effetti collaterali che hanno sfavorito alcune classi in modo sproporzionato. Sarà bene investire per ridurre al minimo gli impatti negativi.

Inoltre, l'IA è diversa da altre tecnologie rivoluzionarie. Migliorare la stampa, l'idraulica, il volo aereo e la telefonia portandole ai loro limiti non comporta il rischio di generare una minaccia alla supremazia del genere umano nel mondo, mentre l'IA può certamente causare una simile minaccia.

In conclusione, l'IA ha fatto grandi progressi in una storia ancora breve, ma l'ultima frase del saggio di Alan Turing (1950) su *Computing Machinery and Intelligence* è valida ancora oggi:

Riusciamo a vedere soltanto un breve tratto di strada avanti a noi, ma ci basta per vedere che molto rimane ancora da fare.

A

- A.1 Analisi di complessità e notazione $O()$
- A.2 Vettori, matrici e algebra lineare
- A.3 Distribuzioni di probabilità
Note storiche e bibliografiche

Basi matematiche

A.1 Analisi di complessità e notazione $O()$

Gli informatici si trovano spesso nelle condizioni di confrontare diversi algoritmi per vedere quanto velocemente si possono eseguire o quanta memoria richiedono. Per far questo ci sono due approcci: il primo consiste nel ricorrere ai **benchmark**, eseguendo cioè gli algoritmi su un computer e misurandone la velocità in secondi e l'occupazione di memoria in byte. Alla fine è questo ciò che conta, ma un benchmark può risultare insoddisfacente a causa della sua specificità: misura solo la prestazione di un particolare programma scritto in un particolare linguaggio e tradotto con un particolare compilatore, in esecuzione su un particolare computer con particolari dati in input. Dal singolo risultato fornito dal benchmark potrebbe essere difficile prevedere come si comporterebbe l'algoritmo con differenti compilatori, computer o insiemi di dati. Il secondo approccio si affida a un'**analisi degli algoritmi** di carattere matematico, indipendentemente dalla particolare implementazione e dal particolare input, come spieghiamo nel seguito.

A.1.1 Analisi asintotica

Esaminiamo l'analisi degli algoritmi mediante il seguente esempio, un programma che calcola la somma di una sequenza di numeri:

```
function SOMMA(sequenza) returns un numero
    somma  $\leftarrow$  0
    for i  $\leftarrow$  1 to LUNGHEZZA(sequenza) do
        somma  $\leftarrow$  somma + sequenza[i]
    return somma
```

Il primo passo dell'analisi è astrarre l'input, ovvero trovare qualche parametro o insieme di parametri che possano caratterizzarne le dimensioni. In questo caso l'input può essere caratterizzato dalla lunghezza della sequenza, che chiameremo n . Il secondo passo è astrarre l'implementazione, per trovare una misura che rispecchia il tempo d'esecuzione dell'algoritmo senza essere legata a un particolare compilatore o computer. Per il programma SOMMA potremmo semplicemente considerare il numero di righe di codice eseguite, o potremmo essere più dettagliati e contare le addizioni, gli assegnamenti, gli accessi agli array e i test condizionali. Entrambi i metodi ci offrono una caratterizzazione che chiameremo $T(n)$, del numero totale di passi effettuati dall'algoritmo in funzione delle dimensioni dell'input. Se contiamo le righe di codice, nel nostro esempio abbiamo $T(n) = 2n + 2$.

Se tutti i programmi fossero semplici come SOMMA, l’analisi degli algoritmi sarebbe banale. Due problemi, comunque, la rendono più complicata: prima di tutto, è raro trovare un parametro come n che caratterizza completamente il numero di passi eseguiti da un algoritmo. Quello che di solito possiamo fare è calcolare il caso pessimo $T_{\text{worst}}(n)$ o quello medio $T_{\text{avg}}(n)$. Per calcolare la media l’analista dovrà formulare un’ipotesi sulla distribuzione degli input.

Il secondo problema è che gli algoritmi tendono a rendere difficile un’analisi esatta: in tal caso è necessario accontentarsi di un’approssimazione. Diciamo che l’algoritmo SOMMA è $O(n)$ per indicare che la sua misura è al più un numero costante di volte n , con possibili eccezioni quando il valore di n è piccolo. Più formalmente,

$$T(n) \text{ è } O(f(n)) \text{ se } T(n) \leq kf(n) \text{ per qualche } k, \text{ per ogni } n > n_0.$$

La notazione $O()$ ci permette di esprimere quella che chiamiamo **analisi asintotica**. Possiamo affermare senza dubbio che, quando n tende asintoticamente all’infinito, un algoritmo $O(n)$ è meglio di un algoritmo $O(n^2)$. Quest’affermazione ovviamente non potrebbe mai essere supportata da un singolo benchmark.

La notazione $O()$ astrae dai fattori costanti, il che la rende più facile da usare ma meno precisa di $T()$. Ad esempio, un algoritmo $O(n^2)$ sarà sempre peggio di un $O(n)$ a lungo termine, ma se i due algoritmi sono $T(n^2 + 1)$ e $T(100n + 1000)$ allora l’algoritmo $O(n^2)$ sarà effettivamente preferibile per $n < 110$.

Nonostante questa limitazione, l’analisi asintotica è lo strumento più diffuso per l’analisi degli algoritmi: proprio perché astrae dall’esatto numero delle operazioni svolte (ignorando il fattore costante k) e il contenuto preciso dell’input (considerando solo la sua dimensione n) l’analisi diventa matematicamente gestibile. La notazione $O()$ è un buon compromesso tra precisione e facilità di analisi.

analisi asintotica

analisi
di complessità

P

NP

NP-completo

A.1.2 NP e problemi intrinsecamente difficili

L’analisi degli algoritmi e la notazione $O()$ ci permettono di discutere dell’efficienza di un particolare algoritmo: tuttavia, non possono aiutarci a determinare se per gestire il nostro problema ne è disponibile uno migliore. L’**analisi di complessità** prende in esame i problemi anziché gli algoritmi. La prima, grande suddivisione si ha tra i problemi che possono essere risolti in tempo polinomiale e quelli per i quali ciò non è possibile, indipendentemente dall’algoritmo usato. La classe dei problemi polinomiali – quelli che si possono risolvere in un tempo $O(n^k)$ per qualche k – è chiamata **P**. Talvolta gli autori chiamano “facili” questi problemi, perché la classe contiene anche i problemi con tempi d’esecuzione $O(\log n)$ e $O(n)$. Tuttavia, può contenere anche problemi di complessità $O(n^{1000})$, ragion per cui è bene non prendere troppo alla lettera il termine “facili”.

Un’altra classe importante di problemi è **NP**, quella dei problemi polinomiali non deterministici. Un problema appartiene a questa classe se qualche algoritmo può ipotizzare una soluzione e poi verificare se una ipotesi è corretta in tempo polinomiale. L’idea è che avendo un numero arbitrariamente grande di processori, in modo da poter provare tutte le soluzioni ipotizzate contemporaneamente, oppure essendo molto fortunati e indovinando sempre la soluzione giusta al primo colpo, i problemi NP diventerebbero P. Una delle questioni aperte più interessanti dell’informatica è se la classe NP sia equivalente alla P quando non si dispone del lusso di un numero infinito di processori o di una infallibile capacità divinatoria. La maggior parte degli informatici è convinta che $P \neq NP$, ovvero che i problemi NP siano intrinsecamente difficili e non ammettano algoritmi polinomiali. Questo, tuttavia, non è mai stato dimostrato.

Coloro che si interessano alla questione se $P = NP$ considerano una sottoclasse di NP, quella dei problemi **NP-completi**. La parola “completi” qui è usata nell’accezione di “più estremi”, e quindi si riferisce ai problemi più difficili della classe NP. È stato dimostrato che ci sono solo due possibilità: o tutti i problemi NP-completi sono in P, o non lo è nessuno.

Questo rende la classe interessante dal punto di vista teorico: ciò non toglie che lo sia anche dal punto di vista pratico, perché un gran numero di problemi reali importanti sono NP-completi. Un esempio è il problema della soddisfacibilità: data una formula della logica proposizionale, esiste un assegnamento dei valori di verità ai simboli proposizionali tale da rendere vera la formula? A meno che non si verifichi un miracolo e sia $P = NP$, non può esistere un algoritmo che risolva *tutti* i problemi di soddisfacibilità in un tempo polinomiale. L'IA, comunque, è più interessata a trovare algoritmi che operano in modo efficiente su problemi *tipici* presi da una distribuzione predeterminata; come abbiamo visto nel Capitolo 7, esistono algoritmi come WALKSAT che si comportano molto bene su un'ampia gamma di problemi.

La classe dei problemi **NP-difficili** è costituita dai problemi riducibili (in tempo polinomiale) a tutti i problemi in NP, perciò, se si risolve un qualsiasi problema NP-difficile, si possono risolvere tutti i problemi in NP. I problemi NP-completi sono tutti NP-difficili, ma esistono alcuni problemi NP-difficili che sono ancora più difficili dei problemi NP-completi.

La classe **co-NP** è il complemento di NP, nel senso che per ogni problema decisionale in NP ce n'è uno corrispondente in co-NP con le risposte "sì" e "no" invertite. Sappiamo che P è un sottoinsieme sia di NP che di co-NP, e si ritiene che esistano problemi in co-NP che non appartengono a P. I problemi **co-NP-completi** sono i più difficili di co-NP.

La classe #P (si dice "numero P" secondo Garey e Johnson (1979), ma molti dicono "diesis P" o "sharp P") è l'insieme di problemi di conteggio (o di calcolo) che corrispondono ai problemi decisionali in NP. I problemi decisionali hanno una risposta nella forma sì/no: c'è una soluzione a questa formula 3-SAT? I problemi di conteggio hanno come risposta un intero: quante soluzioni ammette questa formula 3-SAT? In alcuni casi il problema di conteggio è molto più difficile di quello decisionale. Per esempio, decidere se un grafo bipartito ha un matching perfetto può essere determinato in un tempo $O(VE)$ (dove il grafo ha V vertici ed E archi), ma il problema di conteggio "quanti matching perfetti ha questo grafo bipartito" è #P-completo, il che significa che è difficile quanto ogni altro problema di #P e quindi almeno difficile come un problema NP.

Infine, un'altra classe oggetto di studio è quella dei problemi PSPACE, che richiedono una quantità di spazio polinomiale anche su macchine non deterministiche. Si pensa che i problemi PSPACE-difficili siano peggio di quelli NP-completi, benché non sia impossibile che risulti che $NP = PSPACE$, proprio come potrebbe darsi che $P = NP$.

NP-difficile

co-NP

co-NP-completo

A.2 Vettori, matrici e algebra lineare

I matematici definiscono un **vettore** come un membro di uno spazio vettoriale, ma noi adotteremo una definizione più concreta: un vettore è una sequenza ordinata di valori. Per esempio, in uno spazio bidimensionale avremo vettori come $\mathbf{x} = \langle 3, 4 \rangle$ e $\mathbf{y} = \langle 0, 2 \rangle$. Seguiremo la convenzione di scrivere i nomi dei vettori in grassetto, benché alcuni autori usino invece una freccia o un trattino sopra il nome: \bar{x} o \bar{y} . Si può accedere agli elementi di un vettore usando i pedici: $\mathbf{z} = \langle z_1, z_2, \dots, z_n \rangle$. Un punto potrebbe causare confusione: questo libro sintetizza il lavoro svolto in molti sottocampi dove le sequenze sono chiamate in modo diverso (vettori, liste o tuple) e si utilizzano varie notazioni: $\langle 1,2 \rangle$, $[1, 2]$ o $(1, 2)$.

Le due operazioni fondamentali sono l'addizione tra vettori e la moltiplicazione per uno scalare. L'addizione $\mathbf{x} + \mathbf{y}$ si effettua sommando gli elementi nelle posizioni corrispondenti: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$. Nella moltiplicazione per uno scalare, ogni elemento viene moltiplicato per una costante: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

La lunghezza di un vettore si indica con $|\mathbf{x}|$ e si calcola estraendo la radice quadrata della somma dei quadrati degli elementi: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. Il prodotto scalare $\mathbf{x} \cdot \mathbf{y}$ di due vettori è la somma dei prodotti degli elementi corrispondenti: quindi, $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$; nel nostro caso particolare $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Spesso i vettori sono interpretati come segmenti orientati (frecce) in uno spazio euclideo a n dimensioni. In questo caso la somma di vettori è equivalente a disegnare un vettore con la “coda” in corrispondenza della “testa” dell’altro, mentre il prodotto scalare $\mathbf{x} \cdot \mathbf{y}$ è $|\mathbf{x}| |\mathbf{y}| \cos\theta$, ove θ è l’angolo tra \mathbf{x} e \mathbf{y} .

matrice

Una **matrice** è una griglia rettangolare di valori organizzati per righe e colonne. Quella qui sotto è una matrice \mathbf{A} di dimensioni 3×4 :

$$\begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \end{pmatrix}$$

Il primo indice di $\mathbf{A}_{i,j}$ specifica la riga, il secondo la colonna. Nei linguaggi di programmazione, spesso $\mathbf{A}_{i,j}$ si scrive $\mathbf{A}[i, j]$ o $\mathbf{A}[i][j]$.

La somma di due matrici si ottiene sommando gli elementi corrispondenti; ad esempio $(\mathbf{A} + \mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$. Se \mathbf{A} e \mathbf{B} hanno dimensioni diverse, la somma è indefinita. Possiamo anche definire la moltiplicazione di una matrice per uno scalare: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$. La moltiplicazione tra due matrici è un po’ più complicata. Il prodotto \mathbf{AB} è definito solo se \mathbf{A} ha dimensioni $a \times b$ e \mathbf{B} ha dimensioni $b \times c$ (cioè, la seconda matrice deve avere tante righe quante sono le colonne della prima); il risultato è una matrice di dimensioni $a \times c$. Se le matrici sono di dimensioni appropriate, il risultato è:

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j} \mathbf{B}_{j,k}.$$

La moltiplicazione tra matrici non è commutativa, nemmeno per matrici quadrate: $\mathbf{AB} \neq \mathbf{BA}$ in generale; è però associativa: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Notate che il prodotto scalare può essere espresso come una trasposizione e una moltiplicazione di matrici: $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\top \mathbf{y}$.

matrice identità
matrice trasposta**matrice inversa**
matrice singolare

La **matrice identità** \mathbf{I} ha gli elementi $\mathbf{I}_{i,j}$ pari a uno quando $i = j$ e uguali a 0 nelle altre posizioni. Ha la proprietà che $\mathbf{AI} = \mathbf{A}$ per ogni \mathbf{A} . La matrice **trasposta** di \mathbf{A} , indicata con \mathbf{A}^\top , è costruita scrivendo le righe al posto delle colonne e viceversa: formalmente, $\mathbf{A}^\top_{i,j} = \mathbf{A}_{j,i}$. L'**inversa** di una matrice quadrata \mathbf{A} è un’altra matrice quadrata \mathbf{A}^{-1} tale che $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Per una matrice **singolare**, l’inversa non esiste. Per una matrice non singolare, l’inversa si può calcolare in tempo $O(n^3)$.

Le matrici si usano per risolvere sistemi di equazioni lineari in tempo $O(n^3)$; il tempo è dominato dall’inversione di una matrice di coefficienti. Considerate il seguente insieme di equazioni, per cui vogliamo una soluzione in x, y e z :

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3. \end{aligned}$$

Possiamo rappresentare il sistema con un’equazione matriciale $\mathbf{Ax} = \mathbf{b}$, dove:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}.$$

Per risolvere $\mathbf{Ax} = \mathbf{b}$ moltiplichiamo entrambi i membri per \mathbf{A}^{-1} , ottenendo $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$, che si semplifica in $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Invertendo \mathbf{A} e moltiplicando per \mathbf{b} otteniamo la soluzione:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

Inoltre, indichiamo con $\log(x)$ il logaritmo naturale, $\log_e(x)$, e con $\operatorname{argmax}_x f(x)$ il valore di x per cui $f(x)$ è massima.

A.3 Distribuzioni di probabilità

Una probabilità è una misura su un insieme di eventi che soddisfa tre assiomi.

- La misura di ogni evento è compresa tra 0 e 1. Questo si scrive $0 \leq P(X = x_i) \leq 1$, dove X è una variabile casuale che rappresenta un evento ed x_i sono i possibili valori di X . In generale, le variabili casuali si indicano con lettere maiuscole e i loro valori con lettere minuscole.
- La misura dell'intero insieme è 1: $\sum_{i=1}^n P(X = x_i) = 1$.
- La probabilità dell'unione di eventi disgiunti è pari alla somma delle probabilità dei singoli eventi: $P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$, nel caso in cui x_1, x_2 sono disgiunti.

Un **modello probabilistico** consiste in uno spazio di possibili esiti mutuamente esclusivi insieme alla misura di probabilità associata a ogni esito. Per esempio, in un modello del tempo che farà domani, gli esiti potrebbero essere *sole*, *nuvole*, *pioggia* e *neve*. Un sottoinsieme di questi esiti costituisce un evento. Per esempio, l'evento corrispondente a una precipitazione è il sottoinsieme $\{\text{pioggia, neve}\}$.

modello probabilistico

Usiamo $\mathbf{P}(X)$ per denotare il vettore di valori $\langle P(X = x_1), \dots, P(X = x_n) \rangle$, $P(x_i)$ come abbreviazione di $P(X = x_i)$ e $\sum_x P(x)$ per $\sum_{i=1}^n P(X = x_i)$.

La probabilità condizionale $P(B|A)$ è definita come $P(B \cap A)/P(A)$. A e B sono condizionalmente indipendenti se $P(B|A) = P(B)$ (o, ciò che è equivalente, $P(A|B) = P(A)$).

Nel caso di variabili continue il numero di valori è infinito e, a meno che non ci siano delle cuspidi infinite, la probabilità di un singolo valore esatto è sempre 0. Perciò è più sensato parlare di valore all'interno di un intervallo. Per fare ciò utilizziamo una **funzione di densità di probabilità**, che ha un significato leggermente diverso dalla funzione discreta. Poiché $P(X = x)$ – la probabilità che X abbia esattamente il valore x – è zero, misuriamo invece la probabilità che X cada all'interno di un intervallo intorno a x rispetto all'ampiezza dell'intervallo stesso, e consideriamo il limite al tendere di tale ampiezza a zero:

$$P(x) = \lim_{dx \rightarrow 0} P(x \leq X \leq x + dx)/dx.$$

La funzione di densità non può mai essere negativa, e deve sempre risultare:

$$\int_{-\infty}^{\infty} P(x) dx = 1.$$

Possiamo anche definire la **funzione di distribuzione cumulata** (o cumulativa) $F_X(x)$, che corrisponde alla probabilità che una variabile casuale abbia un valore inferiore a x :

$$F_x(x) = P(X \leq x) = \int_{-\infty}^x P(u) du.$$

funzione di densità di probabilità

funzione di distribuzione cumulata

Notate che la funzione di densità di probabilità è dotata di unità, mentre la funzione di probabilità discreta ne è priva. Ad esempio, se i valori di X sono misurati in secondi, allora la densità si misura in Hz (ovvero, 1/sec). Se i valori di X sono punti in uno spazio tridimensionale misurato in metri, la densità è misurata in $1/m^3$.

Una delle più importanti distribuzioni di probabilità è la **gaussiana**, nota anche come **distribuzione normale**. Utilizziamo la notazione $\mathcal{N}(x; \mu, \sigma^2)$ per indicare la distribuzione normale in funzione di x , con media μ e deviazione standard σ (e quindi varianza σ^2). Tale distribuzione è definita come:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)},$$

distribuzione gaussiana

distribuzione normale standardizzata

distribuzione gaussiana multivariata

dove x è la variabile continua che va da $-\infty$ a $+\infty$. Con media $\mu = 0$ e varianza $\sigma^2 = 1$, otteniamo il caso speciale della **distribuzione normale standardizzata**. Nel caso di un vettore \mathbf{x} a d dimensioni, abbiamo una distribuzione **gaussiana multivariata**:

$$\mathcal{N}(\mathbf{x}; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}((\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu))},$$

dove μ è il vettore media e Σ è la **matrice di covarianza** della distribuzione (vedi sotto). La **distribuzione cumulata** per una distribuzione normale univariata è:

$$F(x) = \int_{-\infty}^x \mathcal{N}(z; \mu, \sigma^2) dz = \frac{1}{2} (1 + \text{erf}(\frac{x-\mu}{\sigma\sqrt{2}})),$$

dove $\text{erf}(x)$ è la cosiddetta **funzione di errore**, che non ha una rappresentazione in forma chiusa.

teorema del limite centrale

Il **teorema del limite centrale** asserisce che la distribuzione formata campionando n variabili casuali indipendenti e prendendo la loro media tende a una distribuzione normale al tendere di n a infinito. Questo vale per quasi ogni collezione di variabili casuali, anche se non sono strettamente indipendenti, a meno che la varianza di un qualsiasi sottoinsieme finito di variabili domini le altre.

valore atteso

Il **valore atteso** (o aspettazione) di una variabile casuale, $E(X)$, è la media o valor medio, pesata con la probabilità di ciascun valore. Per una variabile discreta è:

$$E(X) = \sum_i x_i P(X = x_i).$$

Per una variabile continua, si sostituisce la sommatoria con un integrale e si usa la funzione di densità di probabilità $P(x)$:

$$E(X) = \int_{-\infty}^{\infty} x P(x) dx.$$

Per ogni funzione f abbiamo anche:

$$E(f(X)) = \int_{-\infty}^{\infty} f(x) P(x) dx.$$

Infine, quando necessario, si può specificare la distribuzione per la variabile casuale come pedice per l'operatore di aspettazione:

$$E_{X \sim Q(x)}(g(X)) = \int_{-\infty}^{\infty} g(x) Q(x) dx.$$

varianza

Oltre all'aspettazione, tra le altre importanti proprietà statistiche di una distribuzione ci sono la **varianza**, cioè il valore atteso del quadrato della differenza dalla media μ della distribuzione:

$$\text{Var}(X) = E((X - \mu)^2)$$

deviazione standard

e la **deviazione standard**, cioè la radice quadrata della varianza.

Il **valore quadratico medio** (RMS, *root mean square*) di un insieme di valori (spesso campioni di una variabile casuale) è la radice quadrata della media dei quadrati dei valori:

$$\text{RMS}(x_1, \dots, x_n) = \sqrt{\frac{x_1^2 + \dots + x_n^2}{n}}.$$

La **covarianza** di due variabili casuali è il valore atteso del prodotto delle loro differenze dalle loro medie:

$$\text{cov}(X, Y) = E((X - \mu_X)(Y - \mu_Y)).$$

La **matrice di covarianza**, spesso denotata da Σ , è una matrice di covarianze tra elementi di un vettore di variabili casuali. Dato $\mathbf{X} = \langle X_1, \dots, X_n \rangle^\top$, gli elementi della matrice di covarianza sono i seguenti:

$$\Sigma_{i,j} = \text{cov}(X_i, Y_j) = E((X_i - \mu_i)(X_j - \mu_j)).$$

Con il termine **campionare** da una distribuzione di probabilità si indica l'atto di estrarre un valore a caso. Non sappiamo che cosa ci porterà un'estrazione, ma al limite una grande raccolta di campioni si avvicinerà alla stessa funzione di densità di probabilità della distribuzione da cui è campionata. La **distribuzione uniforme** è una distribuzione in cui tutti gli elementi sono equiprobabili. Perciò, quando diciamo "campionare uniformemente (a caso) dagli interi da 0 a 99" intendiamo estrarre un intero in tale intervallo con la stessa probabilità di tutti gli altri.

matrice di covarianza

campionamento

distribuzione uniforme

Note storiche e bibliografiche

La notazione $O()$, così diffusa nell'informatica, fu introdotta per la prima volta nel contesto della teoria dei numeri dal matematico P. G. H. Bachmann (1894). Il concetto di NP-completezza fu inventato da Cook (1971), e il metodo moderno per ridurre un problema a un altro è dovuto a Karp (1972). Cook e Karp hanno vinto entrambi il premio Turing per il loro lavoro.

Tra i testi sull'analisi e la progettazione degli algoritmi citiamo quelli di Sedgewick e Wayne (2011) e Cormen, Leiserson, Rivest e Stein (2009). Questi libri enfatizzano la progettazione e l'analisi di algoritmi per risolvere problemi trattabili. Per la teoria della NP-completezza e le altre forme di intrattabilità potete far riferimento a Garey e Johnson (1979) o Papadimitriou (1994). Tra i migliori testi sulla probabilità citiamo Chung (1979), Ross (2015), Bertsekas e Tsitsiklis (2008).

B

- B.1 Definire i linguaggi con la forma di Backus-Naur (BNF)
- B.2 Descrivere gli algoritmi con lo pseudocodice

Cenni sui linguaggi e sugli algoritmi

B.1 Definire i linguaggi con la forma di Backus-Naur (BNF)

In questo libro definiamo diversi linguaggi, tra cui la logica proposizionale (Paragrafo 7.4), quella del primo ordine (Figura 8.3) e un sottoinsieme del linguaggio naturale (Capitolo 23 del Volume 2). Un linguaggio formale è definito come un insieme di stringhe, ognuna composta da una sequenza di simboli. Tutti i linguaggi che ci interessano consistono in un insieme infinito di stringhe, che dev'essere caratterizzato in modo conciso: per far questo si usa una **grammatica**. Noi utilizziamo un particolare tipo di grammatica denominato **grammatica libera dal contesto**, perché ogni espressione ha la stessa forma in qualsiasi contesto. Le nostre grammatiche sono tutte scritte nel formalismo chiamato **forma di Backus-Naur (BNF)**. Una grammatica BNF è costituita da quattro componenti.

- Un insieme di **simboli terminali**. Questi sono i simboli o le parole che formano le stringhe del linguaggio, e possono essere lettere (**A, B, C...**) o parole (**a, abaco, abecedario...**) o qualsiasi simbolo appropriato per il dominio.
- Un insieme di **simboli non terminali** che categorizzano sottofrasi del linguaggio. Ad esempio, in italiano il simbolo non terminale *Sintagma Nominale* indica un insieme infinito di stringhe che includono “tu” e “il gran cagnone sbavante”.
- Un **simbolo iniziale**, che è il simbolo non terminale che indica l'insieme completo di stringhe del linguaggio. Per il linguaggio naturale, sarebbe *Frase*; per l'aritmetica, potrebbe essere *Expr*, per i linguaggi di programmazione sarebbe *Programma*.
- Un insieme di **regole di riscrittura**, nella forma *ParteSin* \rightarrow *ParteDex*, dove *ParteSin* è un simbolo non terminale e *ParteDex* una sequenza di zero o più simboli; questi possono essere terminali o non terminali, oppure il simbolo ϵ , che è utilizzato per denotare la stringa vuota.

Una regola di riscrittura della forma:

$$\text{Frase} \rightarrow \text{SintagmaNominale SintagmaVerbale}$$

significa che ogni volta che abbiamo due stringhe categorizzate come un *SintagmaNominale* e un *SintagmaVerbale*, possiamo concatenarle insieme e categorizzare il risultato come una *Frase*. Per brevità, le due regole ($S \rightarrow A$) e ($S \rightarrow B$) si possono scrivere come ($S \rightarrow A | B$). Per illustrare questi concetti, ecco una grammatica BNF per semplici espressioni aritmetiche:

$$\begin{array}{lcl} \text{Expr} & \rightarrow & \text{Expr Operatore Expr} \mid (\text{Expr}) \mid \text{Numero} \\ \text{Numero} & \rightarrow & \text{Cifra} \mid \text{Numero Cifra} \\ \text{Cifra} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \text{Operatore} & \rightarrow & + \mid - \mid \div \mid \times \end{array}$$

I linguaggi e le grammatiche sono trattati più dettagliatamente nel Capitolo 23 del Volume 2. Tenete presente che in altri testi le notazioni BNF potrebbero essere leggermente diverse; potreste avere ad esempio $\langle \text{Cifra} \rangle$ invece di *Cifra* per un simbolo non terminale, ‘parola’ invece di **parola** per un simbolo terminale, $::=$ invece di \rightarrow in una regola.

B.2 Descrivere gli algoritmi con lo pseudocodice

pseudocodice

In questo libro gli algoritmi sono descritti in **pseudocodice**. In gran parte, il nostro pseudocodice dovrebbe essere familiare ai programmati che utilizzano linguaggi come Java, C++ o in particolare Python. In alcuni punti abbiamo usato formule matematiche o linguaggio naturale per descrivere parti che altrimenti sarebbero state scomode da esprimere. Ecco alcuni particolari da tenere presenti.

- **Variabili persistenti.** Usiamo la parola chiave **persistent** per indicare che una variabile riceve un valore iniziale la prima volta che la funzione è invocata e mantiene quel valore (o quello assegnatole in seguito) in tutte le chiamate successive della funzione. In questo le variabili persistenti assomigliano a quelle globali, in quanto il loro ciclo di vita va oltre la singola chiamata, ma sono accessibili solo dall’interno della funzione. I programmi agente nel libro usano variabili persistenti come *memoria*. I programmi che usano variabili persistenti possono essere implementati come *oggetti* in linguaggi object-oriented come C++, Java, Python e Smalltalk. Nei linguaggi funzionali possono essere implementati usando *chiusure funzionali* nell’ambiente che contiene le variabili richieste.
- **Funzioni come valori.** Le funzioni sono indicate in maiuscoletto, mentre le variabili sono scritte in corsivo minuscolo. La maggior parte delle volte, quindi, una chiamata di funzione avrà l’aspetto FUNZIONE(x). Tuttavia, permettiamo che il valore di una variabile sia una funzione; ad esempio, se il valore della variabile f è la funzione radice quadrata, $f(9)$ restituirà 3.
- **L’indentazione è significativa:** l’indentazione è utilizzata per evidenziare lo scope di un ciclo o di una istruzione condizionale, come nei linguaggi Python e CoffeeScript, e a differenza dei linguaggi Java, C++ e Go (che utilizzano parentesi graffe) o Lua e Ruby (che utilizzano **end**).
- **Distrutturazione dell’assegnamento:** la notazione “ $x, y \leftarrow \text{coppia}$ ” significa che il calcolo del membro destro deve fornire una collezione di due elementi, il primo da assegnare a x e il secondo a y . Lo stesso concetto si utilizza in “**for** x, y **in** coppie **do**” e può essere utilizzato per scambiare due variabili: “ $x, y \leftarrow y, x$ ”.
- **Valori di default per i parametri:** la notazione “**function** $F(x, y = 0)$ **returns** un numero” significa che y è un argomento facoltativo con valore di default 0; in pratica, le chiamate $F(3,0)$ e $F(3)$ sono equivalenti.

- **yield**: una funzione che contiene la parola chiave **yield** è un **generatore** che genera una sequenza di valori, uno ogni volta che si incontra l'espressione **yield**. Dopo lo **yield**, la funzione continua l'esecuzione con l'istruzione successiva. I linguaggi Python, Ruby, C# e Javascript (ECMAScript) hanno questa funzionalità. generatore
- **Cicli**: ci sono quattro tipi di cicli:
 - “**for** *x* **in** *c* **do**”: esegue il ciclo con la variabile *x* associata a elementi successivi della collezione *c*;
 - “**for** *i* = 1 **to** *n* **do**”: esegue il ciclo con *i* associata a interi successivi da 1 a *n* incluso;
 - “**while** *condizione* **do**”: significa che la *condizione* viene valutata prima di ogni iterazione del ciclo, e si esce dal ciclo quando la *condizione* è falsa;
 - “**repeat** ... **until** *condizione*”: significa che il ciclo viene eseguito senza condizioni la prima volta, poi viene valutata la *condizione*, e si esce dal ciclo se la *condizione* è vera, altrimenti il ciclo continua a essere eseguito (insieme alla valutazione della condizione alla fine).
- **Liste**: $[x, y, z]$ denota una lista di tre elementi. L'operatore + concatena liste: $[1, 2] + [3, 4] = [1, 2, 3, 4]$. Una lista può essere usata come stack: POP rimuove e restituisce l'ultimo elemento di una lista, TOP restituisce l'ultimo elemento.
- **Insiemi**: $\{x, y, z\}$ denota un insieme di tre elementi. $\{x : p(x)\}$ denota l'insieme di tutti gli elementi *x* per cui *p(x)* è vera.
- **Gli array partono da 1**: il primo indice di un array è 1, come nella consueta notazione matematica (e in R e Julia), e non 0 (come in Python, Java e C).

Bibliografia

Le seguenti abbreviazioni sono utilizzate per conferenze e riviste citate frequentemente:

AAAI	Proceedings of the AAAI Conference on Artificial Intelligence	FGCS	Proceedings of the International Conference on Fifth Generation Computer Systems
AAMAS	Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems	FOCS	Proceedings of the Annual Symposium on Foundations of Computer Science
ACL	Proceedings of the Annual Meeting of the Association for Computational Linguistics	GECCO	Proceedings of the Genetics and Evolutionary Computing Conference
AIJ	Artificial Intelligence (Journal)	HRI	Proceedings of the International Conference on Human-Robot Interaction
AIMag	AI Magazine	ICAPS	Proceedings of the International Conference on Automated Planning and Scheduling
AIPS	Proceedings of the International Conference on AI Planning Systems	ICASSP	Proceedings of the International Conference on Acoustics, Speech, and Signal Processing
AISTATS	Proceedings of the International Conference on Artificial Intelligence and Statistics	ICCV	Proceedings of the International Conference on Computer Vision
BBS	Behavioral and Brain Sciences	ICLP	Proceedings of the International Conference on Logic Programming
CACM	Communications of the Association for Computing Machinery	ICLR	Proceedings of the International Conference on Learning Representations
COGSCI	Proceedings of the Annual Conference of the Cognitive Science Society	ICML	Proceedings of the International Conference on Machine Learning
COLING	Proceedings of the International Conference on Computational Linguistics	ICPR	Proceedings of the International Conference on Pattern Recognition
COLT	Proceedings of the Annual ACM Workshop on Computational Learning Theory	ICRA	Proceedings of the IEEE International Conference on Robotics and Automation
CP	Proceedings of the International Conference on Principles and Practice of Constraint Programming	ICSLP	Proceedings of the International Conference on Speech and Language Processing
CVPR	Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition	IJAR	International Journal of Approximate Reasoning
EC	Proceedings of the ACM Conference on Electronic Commerce	IJCAI	Proceedings of the International Joint Conference on Artificial Intelligence
ECAI	Proceedings of the European Conference on Artificial Intelligence	IJCNN	Proceedings of the International Joint Conference on Neural Networks
ECCV	Proceedings of the European Conference on Computer Vision	IJCV	International Journal of Computer Vision
ECML	Proceedings of the The European Conference on Machine Learning	ILP	Proceedings of the International Workshop on Inductive Logic Programming
ECP	Proceedings of the European Conference on Planning	IROS	Proceedings of the International Conference on Intelligent Robots and Systems
EMNLP	Proceedings of the Conference on Empirical Methods in Natural Language Processing		

ISMIS	Proceedings of the International Symposium on Methodologies for Intelligent Systems	PAMI	IEEE Transactions on Pattern Analysis and Machine Intelligence
ISRR	Proceedings of the International Symposium on Robotics Research	PNAS	Proceedings of the National Academy of Sciences of the United States of America
JACM	Journal of the Association for Computing Machinery	PODS	Proceedings of the ACM International Symposium on Principles of Database Systems
JAIR	Journal of Artificial Intelligence Research	RSS	Proceedings of the Conference on Robotics: Science and Systems
JAR	Journal of Automated Reasoning	SIGIR	Proceedings of the Special Interest Group on Information Retrieval
JASA	Journal of the American Statistical Association	SIGMOD	Proceedings of the ACM SIGMOD International Conference on Management of Data
JMLR	Journal of Machine Learning Research	SODA	Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms
JSL	Journal of Symbolic Logic	STOC	Proceedings of the Annual ACM Symposium on Theory of Computing
KDD	Proceedings of the International Conference on Knowledge Discovery and Data Mining	TARK	Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge
KR	Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning	UAI	Proceedings of the Conference on Uncertainty in Artificial Intelligence
LICS	Proceedings of the IEEE Symposium on Logic in Computer Science		
NeurIPS	Advances in Neural Information Processing Systems		

- Aaronson**, S. (2014). My conversation with “Eugene Goostman”, the chatbot that’s all over the news for allegedly passing the Turing test. Shtetl-Optimized, www.scottaaronson.com/blog/?p=1858.
- Aarts**, E. e Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.
- Ararup**, M., Arentoft, M. M., Parrod, Y., Stader, J. e Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. e Zweben, M. (a cura di), *Knowledge Based Scheduling*. Morgan Kaufmann.
- Abbas**, A. (2018). *Foundations of Multiattribute Utility*. Cambridge University Press.
- Abbeel**, P. e Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML-04*.
- Abney**, S., McAllester, D. A. e Pereira, F. (1999). Relating probabilistic grammars and automata. In *ACL-99*.
- Abramson**, B. (1987). *The expected-outcome model of two-player games*. Ph.D. thesis, Columbia University.
- Abramson**, B. (1990). Expected-outcome: A general model of static evaluation. *PAMI*, 12, 182–193.
- Abreu**, D. e Rubinstein, A. (1988). The structure of Nash equilibrium in repeated games with finite automata. *Econometrica*, 56, 1259–1281.
- Achlioptas**, D. (2009). Random satisfiability. In Biere, A., Heule, M., van Maaren, H. e Walsh, T. (a cura di), *Handbook of Satisfiability*. IOS Press.
- Ackerman**, E. e Guizzo, E. (2016). The next generation of Boston Dynamics’ Atlas robot is quiet, robust e tether free. *IEEE Spectrum*, 24, 2016.
- Ackerman**, N., Freer, C. e Roy, D. (2013). On the computability of conditional probability. arXiv 1005.3014.
- Ackley**, D. H. e Littman, M. L. (1991). Interactions between learning and evolution. In Langton, C., Taylor, C., Farmer, J. D. e Rasmussen, S. (a cura di), *Artificial Life II*. Addison-Wesley.
- Adida**, B. e Birbeck, M. (2008). RDFa primer. Tech. rep., W3C.
- Adolph**, K. E., Kretch, K. S. e LoBue, V. (2014). Fear of heights in infants? *Current Directions in Psychological Science*, 23, 60–66.
- Agerbeck**, C. e Hansen, M. O. (2008). A multiagent approach to solving *NP*-complete problems. Master’s thesis, Technical Univ. of Denmark.
- Aggarwal**, G., Goel, A. e Motwani, R. (2006). Truthful auctions for pricing search keywords. In *EC-06*.
- Agha**, G. (1986). ACTORS: A Model of Concurrent Computation in Distributed Systems. MIT Press.
- Agichtein**, E. e Gravano, L. (2003). Querying text databases for efficient information extraction. In *Proc. IEEE Conference on Data Engineering*.
- Agmon**, S. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6, 382–392.

- Agostinelli**, F., McAleer, S., Shmakov, A. e Baldi, P. (2019). Solving the Rubik's Cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1, 356–363.
- Agrawal**, P., Nair, A. V., Abbeel, P., Malik, J. e Levine, S. (2017). Learning to poke by poking: Experiential learning of intuitive physics. In *NeurIPS 29*.
- Agre**, P. E. e Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *IJCAI-87*.
- Aizerman**, M., Braverman, E. e Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, 821–837.
- Akametalu**, A. K., Fisac, J. F., Gillula, J. H., Kaynama, S., Zeilinger, M. N. e Tomlin, C. J. (2014). Reachability-based safe learning with Gaussian processes. In *53rd IEEE Conference on Decision and Control*.
- Akgun**, B., Cakmak, M., Jiang, K. e Thomaz, A. (2012). Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4, 343–355.
- Aldous**, D. e Vazirani, U. (1994). "Go with the winners" algorithms. In *FOCS-94*.
- Alemi**, A. A., Chollet, F., Een, N., Irving, G., Szegedy, C. e Urban, J. (2017). DeepMath – Deep sequence models for premise selection. In *NeurIPS 29*.
- Allais**, M. (1953). Le comportement de l'homme rationnel devant la risque: critique des postulats et axiomes de l'école Américaine. *Econometrica*, 21, 503–546.
- Allan**, J., Harman, D., Kanoulas, E., Li, D., Van Gysel, C. e Vorhees, E. (2017). Trec 2017 common core track overview. In *Proc. TREC*.
- Allen**, J. F. (1983). Maintaining knowledge about temporal intervals. *CACM*, 26, 832–843.
- Allen**, J. F. (1984). Towards a general theory of action and time. *AI*, 23, 123–154.
- Allen**, J. F. (1991). Time and time again: The many ways to represent time. *Int. J. Intelligent Systems*, 6, 341–355.
- Allen**, J. F., Hender, J. e Tate, A. (a cura di). (1990). *Readings in Planning*. Morgan Kaufmann.
- Allen**, P. e Greaves, M. (2011). The singularity isn't near. *Technology review*, 12, 7–8.
- Allen-Zhu**, Z., Li, Y. e Song, Z. (2018). A convergence theory for deep learning via overparameterization. arXiv:1811.03962.
- Alterman**, R. (1988). Adaptive planning. *Cognitive Science*, 12, 393–422.
- Amarel**, S. (1967). An approach to heuristic problem-solving and theorem proving in the propositional calculus. In Hart, J. e Takasu, S. (a cura di), *Systems and Computer Science*. University of Toronto Press.
- Amarel**, S. (1968). On representations of problems of reasoning about actions. In Michie, D. (a cura di), *Machine Intelligence 3*, Vol. 3. Elsevier.
- Amir**, E. e Russell, S. J. (2003). Logical filtering. In *IJCAI-03*.
- Amit**, Y. e Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9, 1545–1588.
- Amodei**, D. e Hernandez, D. (2018). AI and compute. OpenAI blog, blog.openai.com/ai-and-compute/.
- Amodei**, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J. e Mané, D. (2016). Concrete problems in AI safety. arXiv:1606.06565.
- Andersen**, S. K., Olesen, K. G., Jensen, F. V. e Jensen, F. (1989). HUGIN—A shell for building Bayesian belief universes for expert systems. In *IJCAI-89*.
- Anderson**, J. R. (1980). *Cognitive Psychology and Its Implications*. W. H. Freeman.
- Anderson**, J. R. (1983). *The Architecture of Cognition*. Harvard University Press.
- Anderson**, K., Sturtevant, N. R., Holte, R. C. e Schaeffer, J. (2008). Coarse-to-fine search techniques. Tech. rep., University of Alberta.
- Andoni**, A. e Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS-06*.
- Andor**, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S. e Collins, M. (2016). Globally normalized transition-based neural networks. arXiv:1603.06042.
- Andre**, D., Friedman, N. e Parr, R. (1998). Generalized prioritized sweeping. In *NeurIPS 10*.
- Andre**, D. e Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI-02*.
- Andreae**, P. (1985). *Justified Generalisation: Learning Procedures from Examples*. Ph.D. thesis, MIT.
- Andrieu**, C., Doucet, A. e Holenstein, R. (2010). Particle Markov chain Monte Carlo methods. *J. Royal Statistical Society*, 72, 269–342.
- Andrychowicz**, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018a). Learning dexterous in-hand manipulation. arXiv:1808.00177.
- Andrychowicz**, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P. e Zaremba, W. (2018b). Hindsight experience replay. In *NeurIPS 30*.
- Aneja**, J., Deshpande, A. e Schwing, A. (2018). Convolutional image captioning. In *CVPR-18*.

- Aoki**, M. (1965). Optimal control of partially observable Markov systems. *J. Franklin Institute*, 280, 367–386.
- Appel**, K. e Haken, W. (1977). Every planar map is four colorable: Part I: Discharging. *Illinois J. Math.*, 21, 429–490.
- Appelt**, D. (1999). Introduction to information extraction. *AI Communications*, 12, 161–172.
- Apt**, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science*, 221, 179–210.
- Apt**, K. R. (2003). *Principles of Constraint Programming*. Cambridge University Press.
- Apté**, C., Damerau, F. e Weiss, S. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12, 233–251.
- Arbuthnot**, J. (1692). *Of the Laws of Chance*. Motte, London. Translation into English, with additions, of Huygens (1657).
- Archibald**, C., Altman, A. e Shoham, Y. (2009). Analysis of a winning computational billiards player. In *IJCAI-09*.
- Arfaee**, S. J., Zilles, S. e Holte, R. C. (2010). Bootstrap learning of heuristic functions. In *Third Annual Symposium on Combinatorial Search*.
- Argall**, B. D., Chernova, S., Veloso, M. e Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57, 469–483.
- Ariely**, D. (2009). *Predictably Irrational* (edizione riveduta). Harper.
- Arkin**, R. (1998). *Behavior-Based Robotics*. MIT Press.
- Arkin**, R. (2015). The case for banning killer robots: Counterpoint. *CACM*, 58.
- Armando**, A., Carbone, R., Compagna, L., Cuellar, J. e Tobarra, L. (2008). Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for Google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering*.
- Armstrong**, S. e Levinstein, B. (2017). Low impact artificial intelligences. arXiv:1705.10720.
- Arnauld**, A. (1662). *La logique, ou l'art de penser*. Chez Charles Savreux, Paris.
- Arora**, N. S., Russell, S. J. e Sudderth, E. (2013). NET-VISA: Network processing vertically integrated seismic analysis. *Bull. Seism. Soc. Amer.*, 103, 709–729.
- Arora**, S. (1998). Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *JACM*, 45, 753–782.
- Arpit**, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y. e Lacoste-Julien, S. (2017). A closer look at memorization in deep networks. arXiv:1706.05394.
- Arrow**, K. J. (1951). *Social Choice and Individual Values*. Wiley.
- Arulampalam**, M. S., Maskell, S., Gordon, N. e Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50, 174–188.
- Arulkumaran**, K., Deisenroth, M. P., Brundage, M. e Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34, 26–38.
- Arunachalam**, R. e Sadeh, N. M. (2005). The supply chain trading agent competition. *Electronic Commerce Research and Applications*, Spring, 66–84.
- Ashby**, W. R. (1940). Adaptiveness and equilibrium. *J. Mental Science*, 86, 478–483.
- Ashby**, W. R. (1948). Design for a brain. *Electronic Engineering*, December, 379–383.
- Ashby**, W. R. (1952). *Design for a Brain*. Wiley.
- Asimov**, I. (1942). Runaround. *Astounding Science Fiction*, March.
- Asimov**, I. (1950). *I, Robot*. Doubleday.
- Asimov**, I. (1958). The feeling of power. *If: Worlds of Science Fiction*, February.
- Astrom**, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10, 174–205.
- Atkeson**, C. G., Moore, A. W. e Schaal, S. (1997). Locally weighted learning for control. In *Lazy learning*. Springer.
- Audi**, R. (a cura di). (1999). *The Cambridge Dictionary of Philosophy*. Cambridge University Press.
- Auer**, P., Cesa-Bianchi, N. e Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47, 235–256.
- Aumann**, R. e Brandenburger, A. (1995). Epistemic conditions for nash equilibrium. *Econometrica*, 67, 1161–1180.
- Axelrod**, R. (1985). *The Evolution of Cooperation*. Basic Books.
- Ba**, J. L., Kiros, J. R. e Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450.
- Baader**, F., Calvanese, D., McGuinness, D., Nardi, D. e Patel-Schneider, P. (2007). *The Description Logic Handbook* (2nd edition). Cambridge University Press.

- Baader**, F. e Snyder, W. (2001). Unification theory. In Robinson, J. e Voronkov, A. (a cura di), *Handbook of Automated Reasoning*. Elsevier.
- Bacchus**, F. (1990). *Representing and Reasoning with Probabilistic Knowledge*. MIT Press.
- Bacchus**, F. e Grove, A. (1995). Graphical models for preference and utility. In *UAI-95*.
- Bacchus**, F. e Grove, A. (1996). Utility independence in a qualitative decision theory. In *KR-96*.
- Bacchus**, F., Grove, A., Halpern, J. Y. e Koller, D. (1992). From statistics to beliefs. In *AAAI-92*.
- Bacchus**, F. e van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *AAAI-98*.
- Bacchus**, F. e van Run, P. (1995). Dynamic variable ordering in CSPs. In *CP-95*.
- Bacchus**, F., Dalmao, S. e Pitassi, T. (2003). Value elimination: Bayesian inference via backtracking search. In *UAI-03*.
- Bachmann**, P. G. H. (1894). *Die analytische Zahlentheorie*. B. G. Teubner, Leipzig.
- Backus**, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. *Proc. Int'l Conf. on Information Processing*.
- Bacon**, F. (1609). *Wisdom of the Ancients*. Cassell and Company.
- Baeza-Yates**, R. e Ribeiro-Neto, B. (2011). *Modern Information Retrieval* (2nd edition). Addison-Wesley.
- Bagdasaryan**, E., Veit, A., Hua, Y., Estrin, D. e Shmatikov, V. (2018). How to backdoor federated learning. arXiv:1807.00459.
- Bagnell**, J. A. e Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *ICRA-01*.
- Bahdanau**, D., Cho, K. e Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR-15*.
- Bahubalendruni**, M. R. e Biswal, B. B. (2016). A review on assembly sequence generation and its automation. *Proc. Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230, 824–838.
- Bai**, A. e Russell, S. J. (2017). Efficient reinforcement learning with hierarchies of machines by leveraging internal transitions. In *JCAI-17*.
- Bai**, H., Cai, S., Ye, N., Hsu, D. e Lee, W. S. (2015). Intention-aware online POMDP planning for autonomous driving in a crowd. In *ICRA-15*.
- Bajcsy**, A., Losey, D. P., O'Malley, M. K. e Dragan, A. D. (2017). Learning robot objectives from physical human interaction. *Proceedings of Machine Learning Research*, 78, 217–226.
- Baker**, C. L., Saxe, R. e Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, 113, 329–349.
- Baker**, J. (1975). The Dragon system—An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23, 24–29.
- Baker**, J. (1979). Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*.
- Baldi**, P., Chauvin, Y., Hunkapiller, T. e McClure, M. (1994). Hidden Markov models of biological primary sequence information. *PNAS*, 91, 1059–1063.
- Baldwin**, J. M. (1896). A new factor in evolution. *American Naturalist*, 30, 441–451. Continued on pages 536–553.
- Ballard**, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *AIJ*, 21, 327–350.
- Baluja**, S. (1997). Genetic algorithms and explicit search statistics. In *NeurIPS 9*.
- Bancilhon**, F., Maier, D., Sagiv, Y. e Ullman, J. D. (1986). Magic sets and other strange ways to implement logic programs. In *PODS-86*.
- Banko**, M. e Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *ACL-01*.
- Banko**, M., Brill, E., Dumais, S. T. e Lin, J. (2002). AskMSR: Question answering using the worldwide web. In *Proc. AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*.
- Banko**, M., Cafarella, M. J., Soderland, S., Broadhead, M. e Etzioni, O. (2007). Open information extraction from the web. In *IJCAI-07*.
- Banko**, M. e Etzioni, O. (2008). The tradeoffs between open and traditional relation extraction. In *ACL-08*.
- Bansal**, K., Loos, S., Rabe, M. N., Szegedy, C. e Wilcox, S. (2019). HOList: An environment for machine learning of higher-order theorem proving (extended version). arXiv:1904.03241.
- Bar-Hillel**, Y. (1954). Indexical expressions. *Mind*, 63, 359–379.
- Bar-Shalom**, Y. (a cura di). (1992). *Multitarget-Multisensor Tracking: Advanced Applications*. Artech House.
- Bar-Shalom**, Y. e Fortmann, T. E. (1988). *Tracking and Data Association*. Academic Press.
- Bar-Shalom**, Y., Li, X.-R. e Kirubarajan, T. (2001). *Estimation, Tracking and Navigation: Theory, Algorithms and Software*. Wiley.
- Barber**, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

- Barr**, A. e Feigenbaum, E. A. (a cura di). (1981). *The Handbook of Artificial Intelligence*, Vol. 1. HeurisTech Press and William Kaufmann.
- Barreiro**, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., et al. (2012). EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. *4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*.
- Barreno**, M., Nelson, B., Joseph, A. D. e Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81, 121–148.
- Barrett**, S. e Stone, P. (2015). Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI-15*.
- Barták**, R., Salido, M. A. e Rossi, F. (2010). New trends in constraint satisfaction, planning, and scheduling: A survey. *The Knowledge Engineering Review*, 25, 249–279.
- Bartholdi**, J. J., Tovey, C. A. e Trick, M. A. (1989). The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6, 227–241.
- Barto**, A. G., Bradtke, S. J. e Singh, S. (1995). Learning to act using real-time dynamic programming. *AIJ*, 73, 81–138.
- Barto**, A. G., Sutton, R. S. e Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40, 201–211.
- Barwise**, J. e Etchemendy, J. (2002). *Language, Proof and Logic*. CSLI Press.
- Baum**, E., Boneh, D. e Garrett, C. (1995). On genetic algorithms. In *COLT-95*.
- Baum**, E. e Smith, W. D. (1997). A Bayesian approach to relevance in game playing. *AIJ*, 97, 195–242.
- Baum**, L. E. e Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 41, 1554–1563.
- Baxter**, J. e Bartlett, P. (2000). Reinforcement learning in POMDPs via direct gradient ascent. In *ICML-00*.
- Bayardo**, R. J. e Agrawal, R. (2005). Data privacy through optimal k-anonymization. In *Proc. 21st Int'l Conf. on Data Engineering*.
- Bayardo**, R. J. e Miranker, D. P. (1994). An optimal backtrack algorithm for tree-structured constraint satisfaction problems. *AIJ*, 71, 159–181.
- Bayardo**, R. J. e Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *AAAI-97*.
- Bayes**, T. (1763). An essay towards solving a problem in the doctrine of chances. *Phil. Trans. Roy. Soc.*, 53, 370–418.
- Beal**, J. e Winston, P. H. (2009). The new frontier of human-level artificial intelligence. *IEEE Intelligent Systems*, 24, 21–23.
- Beardon**, A. F., Candeal, J. C., Herden, G., Induráin, E. e Mehta, G. B. (2002). The non-existence of a utility function and the structure of non-representable preference relations. *Journal of Mathematical Economics*, 37, 17 – 38.
- Beattie**, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittweis, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S. e Petersen, S. (2016). DeepMind lab. arXiv:1612.03801.
- Bechhofer**, R. (1954). A single-sample multiple decision procedure for ranking means of normal populations with known variances. *Annals of Mathematical Statistics*, 25, 16–39.
- Beck**, J. C., Feng, T. K. e Watson, J.-P. (2011). Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23, 1–14.
- Beckett**, B. e Posegga, J. (1995). Leantap: Lean, tableau-based deduction. *JAR*, 15, 339–358.
- Beeri**, C., Fagin, R., Maier, D. e Yannakakis, M. (1983). On the desirability of acyclic database schemes. *JACM*, 30, 479–513.
- Bekey**, G. (2008). *Robotics: State Of The Art And Future Challenges*. Imperial College Press.
- Belkin**, M., Hsu, D., Ma, S. e Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off. *PNAS*, 116, 15849–15854.
- Bell**, C. e Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proc. Third Alvey IKBS SIG Workshop*.
- Bell**, J. L. e Machover, M. (1977). *A Course in Mathematical Logic*. Elsevier.
- Bellamy**, E. (2003). *Looking Backward: 2000-1887*. Broadview Press.
- Bellamy**, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K. N., Richards, J. T., Saha, D., Sattigeri, P., Singh, M., Varshney, K. R. e Zhang, Y. (2018). AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. arXiv:1810.01943.

- Bellemare**, M. G., Naddaf, Y., Veness, J. e Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *JAIR*, 47, 253–279.
- Bellman**, R. E. (1952). On the theory of dynamic programming. *PNAS*, 38, 716–719.
- Bellman**, R. E. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16.
- Bellman**, R. E. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.
- Bellman**, R. E. (1965). On the application of dynamic programming to the determination of optimal play in chess and checkers. *PNAS*, 53, 244–246.
- Bellman**, R. E. (1984). *Eye of the Hurricane*. World Scientific.
- Bellman**, R. E. e Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press.
- Bellman**, R. E. (1957). *Dynamic Programming*. Princeton University Press.
- Ben-Tal**, A. e Nemirovski, A. (2001). *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM (Society for Industrial and Applied Mathematics).
- Bengio**, Y., Simard, P. e Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5, 157–166.
- Bengio**, Y. e Bengio, S. (2001). Modeling high-dimensional discrete data with multi-layer neural networks. In *NeurIPS 13*.
- Bengio**, Y., Ducharme, R., Vincent, P. e Jauvin, C. (2003). A neural probabilistic language model. *JMLR*, 3, 1137–1155.
- Bengio**, Y. e LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D. e Weston, J. (a cura di), *Large-Scale Kernel Machines*. MIT Press.
- Benjamin**, M. (2013). *Drone Warfare: Killing by Remote Control*. Verso Books.
- Bentham**, J. (1823). *Principles of Morals and Legislation*. Oxford University Press, Oxford. Original work published in 1789.
- Benzmüller**, C. e Paleo, B. W. (2013). Formalization, mechanization and automation of Gödel's proof of God's existence. arXiv:1308.4526.
- Beresniak**, A., Medina-Lara, A., Auray, J. P., De Wever, A., Praet, J.-C., Tarricone, R., Torbica, A., Dupont, D., Lamure, M. e Duru, G. (2015). Validation of the underlying assumptions of the quality-adjusted life-years outcome: Results from the ECHOOUTCOME European project. *PharmacoEconomics*, 33, 61–69.
- Berger**, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag.
- Bergstra**, J. e Bengio, Y. (2012). Random search for hyper-parameter optimization. *JMLR*, 13, 281–305.
- Berk**, R., Heidari, H., Jabbari, S., Kearns, M. e Roth, A. (2017). Fairness in criminal justice risk assessments: The state of the art. arXiv:1703.09207.
- Berkson**, J. (1944). Application of the logistic function to bio-assay. *JASA*, 39, 357–365.
- Berleur**, J. e Brunnstein, K. (2001). *Ethics of Computing: Codes, Spaces for Discussion and Law*. Chapman and Hall.
- Berlin**, K., Koren, S., Chin, C.-S., Drake, J. P., Landolin, J. M. e Phillippy, A. M. (2015). Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology*, 33, 623.
- Berliner**, H. J. (1979). The B* tree search algorithm: A best-first proof procedure. *AIJ*, 12, 23–40.
- Berliner**, H. J. (1980a). Backgammon computer program beats world champion. *AIJ*, 14, 205–220.
- Berliner**, H. J. (1980b). Computer backgammon. *Scientific American*, 249, 64–72.
- Bermúdez-Chacón**, R., Gonnet, G. H. e Smith, K. (2015). Automatic problem-specific hyperparameter optimization and model selection for supervised machine learning. Tech. rep., ETH Zürich.
- Bernardo**, J. M. e Smith, A. (1994). *Bayesian Theory*. Wiley.
- Berners-Lee**, T., Hendler, J. e Lassila, O. (2001). The semantic web. *Scientific American*, 284, 34–43.
- Bernoulli**, D. (1738). Specimen theoriae novae de mensura sortis. *Proc. St. Petersburg Imperial Academy of Sciences*, 5, 175–192.
- Bernstein**, P. L. (1996). *Against the Gods: The Remarkable Story of Risk*. Wiley.
- Berrada**, L., Zisserman, A. e Kumar, M. P. (2019). Training neural networks for and by interpolation. arXiv:1906.05661.
- Berrou**, C., Glavieux, A. e Thitimajshima, P. (1993). Near Shannon limit error control-correcting coding and decoding: Turbo-codes. 1. In *Proc. IEEE International Conference on Communications*.
- Berry**, D. A. e Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall.
- Bertele**, U. e Brioschi, F. (1972). *Nonserial Dynamic Programming*. Academic Press.
- Bertoli**, P., Cimatti, A. e Roveri, M. (2001a). Heuristic search + symbolic model checking = efficient conformant planning. In *IJCAI-01*.
- Bertoli**, P., Cimatti, A., Roveri, M. e Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI-01*.

- Bertot**, Y., Casteran, P., Huet, G. e Paulin-Mohring, C. (2004). *Interactive Theorem Proving and Program Development*. Springer.
- Bertsekas**, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall.
- Bertsekas**, D. e Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.
- Bertsekas**, D. e Tsitsiklis, J. N. (2008). *Introduction to Probability* (2nd edition). Athena Scientific.
- Bertsekas**, D. e Shreve, S. E. (2007). *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific.
- Bertsimas**, D., Delarue, A. e Martin, S. (2019). Optimizing schools' start time and bus routes. *PNAS*, 116(13), 5943–5948.
- Bertsimas**, D. e Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106, 1039–1082.
- Bessen**, J. (2015). *Learning by Doing: The Real Connection between Innovation, Wages, and Wealth*. Yale University Press.
- Bessière**, C. (2006). Constraint propagation. In Rossi, F., van Beek, P. e Walsh, T. (a cura di), *Handbook of Constraint Programming*. Elsevier.
- Beutel**, A., Chen, J., Doshi, T., Qian, H., Woodruff, A., Luu, C., Kreitmann, P., Bischof, J. e Chi, E. H. (2019). Putting fairness principles into practice: Challenges, metrics, and improvements. arXiv:1901.04562.
- Bhar**, R. e Hamori, S. (2004). *Hidden Markov Models: Applications to Financial Economics*. Springer.
- Bibel**, W. (1993). *Deduction: Automated Logic*. Academic Press.
- Bien**, J., Tibshirani, R., et al. (2011). Prototype selection for interpretable classification. *Annals of Applied Statistics*, 5, 2403–2424.
- Biere**, A., Heule, M., van Maaren, H. e Walsh, T. (a cura di). (2009). *Handbook of Satisfiability*. IOS Press.
- Bies**, A., Mott, J. e Warner, C. (2015). English news text treebank: Penn treebank revised. Linguistic Data Consortium.
- Billings**, D., Burch, N., Davidson, A., Holte, R. C., Schaeffer, J., Schauenberg, T. e Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI-03*.
- Billingsley**, P. (2012). *Probability and Measure* (4th edition). Wiley.
- Binder**, J., Koller, D., Russell, S. J. e Kanazawa, K. (1997a). Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29, 213–244.
- Binder**, J., Murphy, K. e Russell, S. J. (1997b). Space-efficient inference in dynamic probabilistic networks. In *IJCAI-97*.
- Bingham**, E., Chen, J., Jankowiak, M., Obermeyer, F., Pradhan, N., Karlaftos, T., Singh, R., Szerlip, P., Horsfall, P. e Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *JMLR*, 20, 1–26.
- Binmore**, K. (1982). *Essays on Foundations of Game Theory*. Pitman.
- Biran**, O. e Cotton, C. (2017). Explanation and justification in machine learning: A survey. In *Proc. IJCAI-17 Workshop on Explainable AI*.
- Bishop**, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.
- Bishop**, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer-Verlag.
- Bisson**, T. (1990). They're made out of meat. *Omni Magazine*.
- Bistarelli**, S., Montanari, U. e Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *JACM*, 44, 201–236.
- Bitner**, J. R. e Reingold, E. M. (1975). Backtrack programming techniques. *CACM*, 18, 651–656.
- Bizer**, C., Auer, S., Kobilarov, G., Lehmann, J. e Cyganiak, R. (2007). DBpedia – querying Wikipedia like a database. In *16th International Conference on World Wide Web*.
- Blazewicz**, J., Ecker, K., Pesch, E., Schmidt, G. e Weglarz, J. (2007). *Handbook on Scheduling: Models and Methods for Advanced Planning*. Springer-Verlag.
- Blei**, D. M., Ng, A. Y. e Jordan, M. I. (2002). Latent Dirichlet allocation. In *NeurIPS 14*.
- Bliss**, C. I. (1934). The method of probits. *Science*, 79, 38–39.
- Block**, H. D., Knight, B. e Rosenblatt, F. (1962). Analysis of a four-layer series-coupled perceptron. *Rev. Modern Physics*, 34, 275–282.
- Block**, N. (2009). Comparing the major theories of consciousness. In Gazzaniga, M. S. (a cura di), *The Cognitive Neurosciences*. MIT Press.
- Blum**, A. L. e Furst, M. (1997). Fast planning through planning graph analysis. *AIJ*, 90, 281–300.
- Blum**, A. L. (1996). On-line algorithms in machine learning. In *Proc. Workshop on On-Line Algorithms, Dagstuhl*.
- Blum**, A. L., Hopcroft, J. e Kannan, R. (2020). *Foundations of Data Science*. Cambridge University Press.
- Blum**, A. L. e Mitchell, T. M. (1998). Combining labeled and unlabeled data with co-training. In *COLT-98*.
- Blumer**, A., Ehrenfeucht, A., Haussler, D. e Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 36, 929–965.

- Bobrow, D. G.** (1967). Natural language input for a computer problem solving system. In Minsky, M. L. (a cura di), *Semantic Information Processing*. MIT Press.
- Bod, R.** (2008). The data-oriented parsing approach: Theory and application. In *Computational Intelligence: A Compendium*. Springer-Verlag.
- Bod, R., Scha, R. e Sima'an, K.** (2003). *Data-Oriented Parsing*. CSLI Press.
- Boddington, P.** (2017). *Towards a Code of Ethics for Artificial Intelligence*. Springer-Verlag.
- Boden, M. A.** (a cura di). (1990). *The Philosophy of Artificial Intelligence*. Oxford University Press.
- Bolognesi, A. e Ciancarini, P.** (2003). Computer programming of kriegspiel endings: The case of KR vs. K. In *Advances in Computer Games 10*.
- Bolton, R. J. e Hand, D. J.** (2002). Statistical fraud detection: A review. *Statistical science*, 17, 235–249.
- Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A. e Seth, K.** (2017). Practical secure aggregation for privacy-preserving machine learning. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*.
- Bond, A. H. e Gasser, L.** (a cura di). (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann.
- Bonet, B.** (2002). An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *ICML-02*.
- Bonet, B. e Geffner, H.** (1999). Planning as heuristic search: New results. In *ECP-99*.
- Bonet, B. e Geffner, H.** (2000). Planning with incomplete information as heuristic search in belief space. In *ICAPS-00*.
- Bonet, B. e Geffner, H.** (2005). An algorithm better than AO*? In *AAAI-05*.
- Boole, G.** (1847). *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, and Macmillan.
- Booth, T. L.** (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*.
- Borel, E.** (1921). La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 173, 1304–1308.
- Borenstein, J., Everett, B. e Feng, L.** (1996). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd.
- Borenstein, J. e Koren., Y.** (1991). The vector field histogram—Fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7, 278–288.
- Borgida, A., Brachman, R. J., McGuinness, D. e Alperin Resnick, L.** (1989). CLASSIC: A structural data model for objects. *SIGMOD Record*, 18, 58–67.
- Boroditsky, L.** (2003). Linguistic relativity. In Nadel, L. (a cura di), *Encyclopedia of Cognitive Science*. Macmillan.
- Boser, B., Guyon, I. e Vapnik, V. N.** (1992). A training algorithm for optimal margin classifiers. In *COLT-92*.
- Bosse, M., Newman, P., Leonard, J., Soika, M., Feiten, W. e Teller, S.** (2004). Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework. *Int. J. Robotics Research*, 23, 1113–1139.
- Bostrom, N.** (2005). A history of transhumanist thought. *Journal of Evolution and Technology*, 14, 1–25.
- Bostrom, N.** (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.
- Bottou, L. e Bousquet, O.** (2008). The tradeoffs of large scale learning. In *NeurIPS 20*.
- Bottou, L., Curtis, F. E. e Nocedal, J.** (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60, 223–311.
- Boué, L.** (2019). Real numbers, data science and chaos: How to fit any dataset with a single parameter. arXiv:1904.12320.
- Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S. e Vanhoucke, V.** (2017). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. arXiv:1709.07857.
- Boutilier, C.** (2002). A POMDP formulation of preference elicitation problems. In *AAAI-02*.
- Boutilier, C. e Brafman, R. I.** (2001). Partial-order planning with concurrent interacting actions. *JAIR*, 14, 105–136.
- Boutilier, C., Dearden, R. e Goldszmidt, M.** (2000). Stochastic dynamic programming with factored representations. *AIJ*, 121, 49–107.
- Boutilier, C., Reiter, R. e Price, B.** (2001). Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*.
- Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H. e Poole, D.** (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR*, 21, 135–191.
- Boutilier, C., Friedman, N., Goldszmidt, M. e Koller, D.** (1996). Context-specific independence in Bayesian networks. In *UAI-96*.
- Bouzy, B. e Cazenave, T.** (2001). Computer Go: An AI oriented survey. *AIJ*, 132, 39–103.

- Bowling**, M., Burch, N., Johanson, M. e Tammelin, O. (2015). Heads-up limit hold'em poker is solved. *Science*, 347, 145–149.
- Bowling**, M., Johanson, M., Burch, N. e Szafron, D. (2008). Strategy evaluation in extensive games with importance sampling. In *ICML-08*.
- Bowman**, S., Angeli, G., Potts, C. e Manning, C. (2015). A large annotated corpus for learning natural language inference. In *EMNLP-15*.
- Box**, G. E. P. (1957). Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics*, 6, 81–101.
- Box**, G. E. P., Jenkins, G., Reinsel, G. e Ljung, G. M. (2016). *Time Series Analysis: Forecasting and Control* (5th edition). Wiley.
- Box**, G. E. P. e Tiao, G. C. (1973). *Bayesian Inference in Statistical Analysis*. Addison-Wesley.
- Boyan**, J. A. e Moore, A. W. (1998). Learning evaluation functions for global optimization and Boolean satisfiability. In *AAAI-98*.
- Boyd**, S. e Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.
- Boyden**, X., Friedman, N. e Koller, D. (1999). Discovering the hidden structure of complex dynamic systems. In *UAI-99*.
- Boyer**, R. S. e Moore, J. S. (1979). *A Computational Logic*. Academic Press.
- Boyer**, R. S. e Moore, J. S. (1984). Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, 91, 181–189.
- Brachman**, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (a cura di), *Associative Networks: Representation and Use of Knowledge by Computers*. Academic Press.
- Brachman**, R. J. e Levesque, H. J. (a cura di). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann.
- Bradt**, R. N., Johnson, S. M. e Karlin, S. (1956). On sequential designs for maximizing the sum of n observations. *Ann. Math. Statist.*, 27, 1060–1074.
- Brafman**, O. e Brafman, R. (2009). *Sway: The Irresistible Pull of Irrational Behavior*. Broadway Business.
- Brafman**, R. I. e Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS-08*.
- Brafman**, R. I. e Tennenholz, M. (2000). A near optimal polynomial time algorithm for learning in certain classes of stochastic games. *AIJ*, 121, 31–47.
- Braitenberg**, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.
- Brandt**, F., Conitzer, V., Endriss, U., Lang, J. e Procaccia, A. D. (a cura di). (2016). *Handbook of Computational Social Choice*. Cambridge University Press.
- Brants**, T. (2000). TnT: A statistical part-of-speech tagger. In *Proc. Sixth Conference on Applied Natural Language Processing*.
- Brants**, T., Popat, A. C., Xu, P., Och, F. J. e Dean, J. (2007). Large language models in machine translation. In *EMNLP-CoNLL-07*.
- Bratko**, I. (2009). *Prolog Programming for Artificial Intelligence* (4th edition). Addison-Wesley.
- Bratman**, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press.
- Breck**, E., Cai, S., Nielsen, E., Salib, M. e Sculley, D. (2016). What's your ML test score? A rubric for ML production systems. In *Proc. NIPS 2016 Workshop on Reliable Machine Learning in the Wild*.
- Breese**, J. S. (1992). Construction of belief and decision networks. *Computational Intelligence*, 8, 624–647.
- Breese**, J. S. e Heckerman, D. (1996). Decision-theoretic troubleshooting: A framework for repair and experiment. In *UAI-96*.
- Breiman**, L., Friedman, J., Olshen, R. A. e Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.
- Breiman**, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.
- Brelaz**, D. (1979). New methods to color the vertices of a graph. *CACM*, 22, 251–256.
- Brent**, R. P. (1973). *Algorithms for Minimization without Derivatives*. Prentice-Hall.
- Bresnan**, J. (1982). *The Mental Representation of Grammatical Relations*. MIT Press.
- Brewka**, G., Dix, J. e Konolige, K. (1997). *Nonmonotonic Reasoning: An Overview*. Center for the Study of Language and Information (CSLI).
- Brickley**, D. e Guha, R. V. (2004). RDF vocabulary description language 1.0: RDF schema. Tech. rep., W3C.
- Briggs**, R. (1985). Knowledge representation in Sanskrit and artificial intelligence. *AIMag*, 6, 32–39.
- Brill**, E. (1992). A simple rule-based part of speech tagger. In *Proc. Third Conference on Applied Natural Language Processing*.
- Brin**, D. (1998). *The Transparent Society*. Perseus.

- Brin**, S. e Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Proc. Seventh World Wide Web Conference*.
- Bringsjord**, S. (2008). If I were judge. In Epstein, R., Roberts, G. e Beber, G. (a cura di), *Parsing the Turing Test*. Springer.
- Broadbent**, D. E. (1958). *Perception and Communication*. Pergamon.
- Brockman**, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. e Zaremba, W. (2016). OpenAI gym. arXiv:1606.01540.
- Brooks**, R. A. (1986). A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, 2, 14–23.
- Brooks**, R. A. (1989). Engineering approach to building complete, intelligent beings. *Proc. SPIE—the International Society for Optical Engineering*, 1002, 618–625.
- Brooks**, R. A. (1991). Intelligence without representation. *AIJ*, 47, 139–159.
- Brooks**, R. A. e Lozano-Perez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man and Cybernetics*, 15, 224–233.
- Brooks**, R. A. (2017). The seven deadly sins of AI predictions. *MIT Technology Review*, Oct 6.
- Brooks**, S., Gelman, A., Jones, G. e Meng, X.-L. (2011). *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC.
- Brown**, C., Finkelstein, L. e Purdom, P. (1988). Backtrack searching in the presence of symmetry. In Mora, T. (a cura di), *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Springer-Verlag.
- Brown**, K. C. (1974). A note on the apparent bias of net revenue estimates. *J. Finance*, 29, 1215–1216.
- Brown**, N. e Sandholm, T. (2017). Libratus: The superhuman AI for no-limit poker. In *IJCAI-17*.
- Brown**, N. e Sandholm, T. (2019). Superhuman AI for multiplayer poker. *Science*, 365, 885–890.
- Brown**, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Mercer, R. L. e Roossin, P. (1988). A statistical approach to language translation. In *COLING-88*.
- Brown**, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D. e Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4).
- Browne**, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Liebana, D. P., Samothrakis, S. e Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4, 1–43.
- Brownston**, L., Farrell, R., Kant, E. e Martin, N. (1985). *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley.
- Bruce**, V., Green, P. e Georgeson, M. (2003). *Visual Perception: Physiology, Psychology and Ecology*. Routledge and Kegan Paul.
- Brügmann**, B. (1993). Monte Carlo Go. Tech. rep., Department of Physics, Syracuse University.
- Bryce**, D. e Kambhampati, S. (2007). A tutorial on planning graph-based reachability heuristics. *AIMag*, Spring, 47–83.
- Bryce**, D., Kambhampati, S. e Smith, D. E. (2006). Planning graph heuristics for belief space search. *JAIR*, 26, 35–99.
- Brynjolfsson**, E. e McAfee, A. (2011). *Race Against the Machine*. Digital Frontier Press.
- Brynjolfsson**, E. e McAfee, A. (2014). *The Second Machine Age*. W. W. Norton.
- Brynjolfsson**, E., Rock, D. e Syverson, C. (2018). Artificial intelligence and the modern productivity paradox: A clash of expectations and statistics. In Agrawal, A., Gans, J. e Goldfarb, A. (a cura di), *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press.
- Bryson**, A. E. e Ho, Y.-C. (1969). *Applied Optimal Control*. Blaisdell.
- Bryson**, A. E. (1962). A gradient method for optimizing multi-stage allocation processes. In *Proc. of a Harvard Symposium on Digital Computers and Their Applications*.
- Bryson**, J. J. (2012). A role for consciousness in action selection. *International Journal of Machine Consciousness*, 4, 471–482.
- Bryson**, J. J. e Winfield, A. (2017). Standardizing ethical design for artificial intelligence and autonomous systems. *Computer*, 50, 116–119.
- Buchanan**, B. G., Mitchell, T. M., Smith, R. G. e Johnson, C. R. (1978). Models of learning systems. In *Encyclopedia of Computer Science and Technology*, Vol. 11. Dekker.
- Buchanan**, B. G. e Shortliffe, E. H. (a cura di). (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.
- Buchanan**, B. G., Sutherland, G. L. e Feigenbaum, E. A. (1969). Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D. e Swann, M. (a cura di), *Machine Intelligence 4*. Edinburgh University Press.

- Buck**, C., Heafield, K. e Van Ooyen, B. (2014). N-gram counts and language models from the common crawl. In *Proc. International Conference on Language Resources and Evaluation*.
- Buehler**, M., Iagnemma, K. e Singh, S. (a cura di). (2006). *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer-Verlag.
- Buffon**, G. (1777). *Essai d'arithmetique morale*. Supplement to *Histoire naturelle*, vol. IV.
- Bunt**, H. C. (1985). The formal representation of (quasi-) continuous concepts. In Hobbs, J. R. e Moore, R. C. (a cura di), *Formal Theories of the Commonsense World*. Ablex.
- Buolamwini**, J. e Gebru, T. (2018). Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*.
- Burgard**, W., Cremers, A. B., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W. e Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *AIJ*, 114, 3–55.
- Burkov**, A. (2019). *The Hundred-Page Machine Learning Book*. Burkov.
- Burns**, E., Hatem, M., Leighton, M. J. e Ruml, W. (2012). Implementing fast heuristic search code. In *Symposium on Combinatorial Search*.
- Buro**, M. (1995). ProbCut: An effective selective extension of the alpha-beta algorithm. *J. International Computer Chess Association*, 18, 71–76.
- Buro**, M. (2002). Improving heuristic mini-max search by supervised learning. *AIJ*, 134, 85–99.
- Burstein**, J., Leacock, C. e Swartz, R. (2001). Automated evaluation of essays and short answers. In *Fifth International Computer Assisted Assessment Conference*.
- Burton**, R. (2009). *On Being Certain: Believing You Are Right Even When You're Not*. St. Martin's Griffin.
- Buss**, D. M. (2005). *Handbook of Evolutionary Psychology*. Wiley.
- Butler**, S. (1863). Darwin among the machines. *The Press (Christchurch, New Zealand)*, June 13.
- Bylander**, T. (1994). The computational complexity of propositional STRIPS planning. *AIJ*, 69, 165–204.
- Byrd**, R. H., Lu, P., Nocedal, J. e Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16, 1190–1208.
- Cabeza**, R. e Nyberg, L. (2001). Imaging cognition II: An empirical review of 275 PET and fMRI studies. *J. Cognitive Neuroscience*, 12, 1–47.
- Cafarella**, M. J., Halevy, A., Zhang, Y., Wang, D. Z. e Wu, E. (2008). Webtables: Exploring the power of tables on the web. In *Vldb-08*.
- Calvanese**, D., Lenzerini, M. e Nardi, D. (1999). Unifying class-based representation formalisms. *JAIR*, 11, 199–240.
- Camacho**, R. e Michie, D. (1995). Behavioral cloning: A correction. *AIMag*, 16, 92.
- Campbell**, D. E. e Kelly, J. (2002). Impossibility theorems in the Arrowian framework. In Arrow, K. J., Sen, A. K. e Suzumura, K. (a cura di), *Handbook of Social Choice and Welfare Volume 1*. Elsevier Science.
- Campbell**, M. S., Hoane, A. J. e Hsu, F.-H. (2002). Deep Blue. *AIJ*, 134, 57–83.
- Cannings**, C., Thompson, E. e Skolnick, M. H. (1978). Probability functions on complex pedigrees. *Advances in Applied Probability*, 10, 26–61.
- Canny**, J. e Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *FOCS-87*.
- Canny**, J. (1986). A computational approach to edge detection. *PAMI*, 8, 679–698.
- Canny**, J. (1988). *The Complexity of Robot Motion Planning*. MIT Press.
- Capen**, E., Clapp, R. e Campbell, W. (1971). Competitive bidding in high-risk situations. *J. Petroleum Technology*, 23, 641–653.
- Carbonell**, J. G. (1983). Derivational analogy and its role in problem solving. In *AAAI-83*.
- Carbonell**, J. G., Knoblock, C. A. e Minton, S. (1989). PRODIGY: An integrated architecture for planning and learning. Technical report, Computer Science Department, Carnegie-Mellon University.
- Carbonnel**, C. e Cooper, M. C. (2016). Tractability in constraint satisfaction problems: A survey. *Constraints*, 21(2), 115–144.
- Cardano**, G. (1663). *Liber de ludo aleae*. Lyons.
- Carlini**, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A. e Kurakin, A. (2019). On evaluating adversarial robustness. arXiv:1902.06705.
- Carnap**, R. (1928). *Der logische Aufbau der Welt*. Weltkreis-verlag. Translated into English as *The Logical Structure of the World* (Carnap, 1967).
- Carnap**, R. (1948). On the application of inductive logic. *Philosophy and Phenomenological Research*, 8, 133–148.
- Carnap**, R. (1950). *Logical Foundations of Probability*. University of Chicago Press.
- Carpenter**, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P. e Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, 76, 1–32.
- Carroll**, S. (2007). *The Making of the Fittest: DNA and the Ultimate Forensic Record of Evolution*. Norton.

- Casati**, R. e Varzi, A. (1999). *Parts and Places: The Structures of Spatial Representation*. MIT Press.
- Cassandra**, A. R., Kaelbling, L. P. e Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *AAAI-94*.
- Cassandras**, C. G. e Lygeros, J. (2006). *Stochastic Hybrid Systems*. CRC Press.
- Castro**, R., Coates, M., Liang, G., Nowak, R. e Yu, B. (2004). Network tomography: Recent developments. *Statistical Science*, 19, 499–517.
- Cauchy**, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25, 536–538.
- Cesa-Bianchi**, N. e Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press.
- Chajewska**, U., Koller, D. e Parr, R. (2000). Making rational decisions using adaptive utility elicitation. In *AAAI-00*.
- Chakrabarti**, P. P., Ghose, S., Acharya, A. e de Sarkar, S. C. (1989). Heuristic search in restricted memory. *AIJ*, 41, 197–222.
- Chalkiadakis**, G., Elkind, E. e Wooldridge, M. (2011). *Computational Aspects of Cooperative Game Theory*. Morgan Kaufmann.
- Chalmers**, D. J. (1992). Subsymbolic computation and the Chinese room. In Dinsmore, J. (a cura di), *The symbolic and connectionist paradigms: Closing the gap*. Lawrence Erlbaum.
- Chandola**, V., Banerjee, A. e Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41.
- Chandra**, A. K. e Harel, D. (1980). Computable queries for relational data bases. *J. Computer and System Sciences*, 21, 156–178.
- Chang**, C.-L. e Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.
- Chang**, H. S., Fu, M. C., Hu, J. e Marcus, S. I. (2005). An adaptive sampling algorithm for solving Markov decision processes. *Operations Research*, 53, 126–139.
- Chao**, W.-L., Hu, H. e Sha, F. (2018). Being negative but constructively: Lessons learnt from creating better visual question answering datasets. In *ACL-18*.
- Chapman**, D. (1987). Planning for conjunctive goals. *AIJ*, 32, 333–377.
- Charniak**, E. (1993). *Statistical Language Learning*. MIT Press.
- Charniak**, E. (1996). Tree-bank grammars. In *AAAI-96*.
- Charniak**, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*.
- Charniak**, E. e Goldman, R. (1992). A Bayesian model of plan recognition. *AIJ*, 64, 53–79.
- Charniak**, E., Riesbeck, C., McDermott, D. e Meehan, J. (1987). *Artificial Intelligence Programming* (2nd edition). Lawrence Erlbaum.
- Charniak**, E. (1991). Bayesian networks without tears. *AI Mag*, 12, 50–63.
- Charniak**, E. (2018). *Introduction to Deep Learning*. MIT Press.
- Chaslot**, G., Bakkes, S., Szita, I. e Spronck, P. (2008). Monte-Carlo tree search: A new framework for game AI. In *Proc. Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*.
- Chater**, N. e Oaksford, M. (a cura di). (2008). *The Probabilistic Mind: Prospects for Bayesian Cognitive Science*. Oxford University Press.
- Chatfield**, C. (1989). *The Analysis of Time Series: An Introduction* (4th edition). Chapman and Hall.
- Chavira**, M. e Darwiche, A. (2008). On probabilistic inference by weighted model counting. *AIJ*, 172, 772–799.
- Chawla**, N. V., Bowyer, K. W., Hall, L. O. e Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *JAIR*, 16, 321–357.
- Cheeseman**, P. (1985). In defense of probability. In *IJCAI-85*.
- Cheeseman**, P. (1988). An inquiry into computer understanding. *Computational Intelligence*, 4, 58–66.
- Cheeseman**, P., Kanefsky, B. e Taylor, W. (1991). Where the really hard problems are. In *IJCAI-91*.
- Cheeseman**, P., Self, M., Kelly, J. e Stutz, J. (1988). Bayesian classification. In *AAAI-88*.
- Cheeseman**, P. e Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In Fayyad, U., Piatesky-Shapiro, G., Smyth, P. e Uthurusamy, R. (a cura di), *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press.
- Chen**, D. e Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *EMNLP-14*.
- Chen**, J., Holte, R. C., Zilles, S. e Sturtevant, N. R. (2017). Front-to-end bidirectional heuristic search with near-optimal node expansions. *IJCAI-17*.
- Chen**, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Parmar, N., Schuster, M., Chen, Z., Wu, Y. e Hughes, M. (2018). The best of both worlds: Combining recent advances in neural machine translation. In *ACL-18*.
- Chen**, S. F. e Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *ACL-96*.
- Chen**, T. e Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *KDD-16*.

- Cheng**, J. e Druzdzel, M. J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *JAIR*, 13, 155–188.
- Cheng**, J., Greiner, R., Kelly, J., Bell, D. A. e Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *AIJ*, 137, 43–90.
- Chiu**, C., Sainath, T., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R., Rao, K., Gonina, K., Jaity, N., Li, B., Chorowski, J. e Bacchiani, M. (2017). State-of-the-art speech recognition with sequence-to-sequence models. arXiv:1712.01769.
- Chklovski**, T. e Gil, Y. (2005). Improving the design of intelligent acquisition interfaces for collecting world knowledge from web contributors. In *Proc. Third International Conference on Knowledge Capture*.
- Chollet**, F. (2019). On the measure of intelligence. arXiv:1911.01547.
- Chollet**, F. (2017). *Deep Learning with Python*. Manning.
- Chomsky**, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2, 113–124.
- Chomsky**, N. (1957). *Syntactic Structures*. Mouton.
- Choromanska**, A., Henaff, M., Mathieu, M., Arous, G. B. e LeCun, Y. (2014). The loss surface of multilayer networks. arXiv:1412.0233.
- Choset**, H. (1996). *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology.
- Choset**, H., Hutchinson, S., Lynch, K., Kantor, G., Burgard, W., Kavraki, L. e Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press.
- Chouldechova**, A. (2017). Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5, 153–163.
- Chouldechova**, A. e Roth, A. (2018). The frontiers of fairness in machine learning. arXiv:1810.08810.
- Christian**, B. (2011). *The Most Human Human*. Doubleday.
- Christin**, A., Rosenblat, A. e Boyd, D. (2015). Courts and predictive algorithms. *Data & Civil Rights*.
- Chung**, K. L. (1979). *Elementary Probability Theory with Stochastic Processes* (3rd edition). Springer-Verlag.
- Church**, A. (1936). A note on the Entscheidungsproblem. *JSL*, 1, 40–41 and 101–102.
- Church**, A. (1956). *Introduction to Mathematical Logic*. Princeton University Press.
- Church**, K. (1988). A stochastic parts program and noun phrase parser for unrestricted texts. In *Proc. Second Conference on Applied Natural Language Processing*.
- Church**, K. e Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics*, 8, 139–149.
- Church**, K. (2004). Speech and language processing: Can we use the past to predict the future. In *Proc. Conference on Text, Speech, and Dialogue*.
- Church**, K. e Gale, W. A. (1991). A comparison of the enhanced Good–Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5, 19–54.
- Church**, K. e Hestness, J. (2019). A survey of 25 years of evaluation. *Natural Language Engineering*, 25, 753–767.
- Churchland**, P. M. (2013). *Matter and Consciousness* (3rd edition). MIT Press.
- Ciancarini**, P. e Favini, G. P. (2010). Monte Carlo tree search in Kriegspiel. *AIJ*, 174, 670–684.
- Ciancarini**, P. e Wooldridge, M. (2001). *Agent-Oriented Software Engineering*. Springer-Verlag.
- Cimatti**, A., Roveri, M. e Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI-98*.
- Claret**, G., Rajamani, S. K., Nori, A. V., Gordon, A. D. e Borgström, J. (2013). Bayesian inference using data flow analysis. In *Proc. 9th Joint Meeting on Foundations of Software Engineering*.
- Clark**, A. (1998). *Being There: Putting Brain, Body, and World Together Again*. MIT Press.
- Clark**, A. (2015). *Surfing Uncertainty: Prediction, Action, and the Embodied Mind*. Oxford University Press.
- Clark**, K. L. (1978). Negation as failure. In Gallaire, H. e Minker, J. (a cura di), *Logic and Data Bases*. Plenum.
- Clark**, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C. e Tafjord, O. (2018). Think you have solved question answering? Try ARC, the AI2 reasoning challenge. arXiv:1803.05457.
- Clark**, P., Etzioni, O., Khot, T., Mishra, B. D., Richardson, K., et al. (2019). From ‘F’ to ‘A’ on the NY Regents science exams: An overview of the Aristo project. arXiv:1909.01958.
- Clark**, S. e Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *ACL-04*.
- Clarke**, A. C. (1968). *2001: A Space Odyssey*. Signet.
- Clarke**, E. e Grumberg, O. (1987). Research on automatic verification of finite-state concurrent systems. *Annual Review of Computer Science*, 2, 269–290.
- Clearwater**, S. H. (a cura di). (1996). *Market-Based Control*. World Scientific.
- Clocksin**, W. F. e Mellish, C. S. (2003). *Programming in Prolog* (5th edition). Springer-Verlag.

- Clocksin, W. F.** (2003). *Clause and Effect: Prolog Programming for the Working Programmer*. Springer.
- Coase, R. H.** (1960). The problem of social cost. *Journal of Law and Economics*, pp. 1–44.
- Coates, A., Abbeel, P. e Ng, A. Y.** (2009). Apprenticeship learning for helicopter control. *Association for Computing Machinery*, 52(7).
- Cobham, A.** (1964). The intrinsic computational difficulty of functions. In *Proc. International Congress for Logic, Methodology, and Philosophy of Science*.
- Cohen, P. R.** (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.
- Cohen, P. R. e Levesque, H. J.** (1990). Intention is choice with commitment. *AIJ*, 42, 213–261.
- Cohen, P. R., Morgan, J. e Pollack, M. E.** (1990). *Intentions in Communication*. MIT Press.
- Cohen, P. R. e Perrault, C. R.** (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, 3, 177–212.
- Cohn, A. G., Bennett, B., Gooday, J. M. e Gotts, N.** (1997). RCC: A calculus for region based qualitative spatial reasoning. *GeoInformatica*, 1, 275–316.
- Collin, Z., Dechter, R. e Katz, S.** (1999). Self-stabilizing distributed constraint satisfaction. *Chicago J. of Theoretical Computer Science*, 1999.
- Collins, M.** (1999). *Head-driven Statistical Models for Natural Language Processing*. Ph.D. thesis, University of Pennsylvania.
- Collins, M. e Duffy, K.** (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL-02*.
- Colmerauer, A. e Roussel, P.** (1993). The birth of Prolog. *SIGPLAN Notices*, 28, 37–52.
- Colmerauer, A., Kanoui, H., Pasero, R. e Roussel, P.** (1973). Un système de communication homme– machine en Franc,ais. Rapport, Groupe d’Intelligence Artificielle, Université d’Aix-Marseille II.
- Condon, J. H. e Thompson, K.** (1982). Belle chess hardware. In Clarke, M. R. B. (a cura di), *Advances in Computer Chess 3*. Pergamon.
- Congdon, C. B., Huber, M., Kortenkamp, D., Bidlack, C., Cohen, C., Huffman, S., Koss, F., Raschke, U. e Weymouth, T.** (1992). CARMEL versus Flakey: A comparison of two robots. Tech. rep., American Association for Artificial Intelligence.
- Conlisk, J.** (1989). Three variants on the Allais example. *American Economic Review*, 79, 392–407.
- Connell, J.** (1989). *A Colony Architecture for an Artificial Creature*. Ph.D. thesis, Artificial Intelligence Laboratory, MIT.
- Conway, D. e White, J.** (2012). *Machine Learning for Hackers*. O'Reilly.
- Cook, S. A.** (1971). The complexity of theorem-proving procedures. In *STOC-71*.
- Cook, S. A. e Mitchell, D.** (1997). Finding hard instances of the satisfiability problem: A survey. In Du, D., Gu, J. e Pardalos, P. (a cura di), *Satisfiability problems: Theory and applications*. American Mathematical Society.
- Cooper, G.** (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *AIJ*, 42, 393–405.
- Cooper, G. e Herskovits, E.** (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9, 309–347.
- Copeland, J.** (1993). *Artificial Intelligence: A Philosophical Introduction*. Blackwell.
- Corbett-Davies, S. e Goel, S.** (2018). The measure and mismeasure of fairness: A critical review of fair machine learning. arXiv:1808.00023.
- Corbett-Davies, S., Pierson, E., Feller, A., Goel, S. e Huq, A.** (2017). Algorithmic decision making and the cost of fairness. arXiv:1701.08230.
- Cormen, T. H., Leiserson, C. E., Rivest, R. e Stein, C.** (2009). *Introduction to Algorithms* (3rd edition). MIT Press.
- Cortes, C. e Vapnik, V. N.** (1995). Support vector networks. *Machine Learning*, 20, 273–297.
- Cournot, A.** (a cura di). (1838). *Recherches sur les principes mathématiques de la théorie des richesses*. L. Hachette, Paris.
- Cover, T. e Thomas, J.** (2006). *Elements of Information Theory* (2nd edition). Wiley.
- Cowan, J. D. e Sharp, D. H.** (1988a). Neural nets. *Quarterly Reviews of Biophysics*, 21, 365–427.
- Cowan, J. D. e Sharp, D. H.** (1988b). Neural nets and artificial intelligence. *Daedalus*, 117, 85–121.
- Cowell, R., Dawid, A. P., Lauritzen, S. e Spiegelhalter, D. J.** (2002). *Probabilistic Networks and Expert Systems*. Springer.
- Cox, I.** (1993). A review of statistical data association techniques for motion correspondence. *IJCV*, 10, 53–66.
- Cox, I. e Hingorani, S. L.** (1994). An efficient implementation and evaluation of Reid’s multiple hypothesis tracking algorithm for visual tracking. In *ICPR-94*.
- Cox, I. e Wilfong, G. T.** (a cura di). (1990). *Autonomous Robot Vehicles*. Springer Verlag.
- Cox, R. T.** (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14, 1–13.
- Craig, J.** (1989). *Introduction to Robotics: Mechanics and Control* (2nd edition). Addison-Wesley.

- Craik**, K. (1943). *The Nature of Explanation*. Cambridge University Press.
- Cramton**, P., Shoham, Y. e Steinberg, R. (a cura di). (2006). *Combinatorial Auctions*. MIT Press.
- Craven**, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T. M., Nigam, K. e Slattery, S. (2000). Learning to construct knowledge bases from the World Wide Web. *AIJ*, 118, 69–113.
- Crawford**, J. M. e Auton, L. D. (1993). Experimental results on the crossover point in satisfiability problems. In *AAAI-93*.
- Crick**, F. (1999). The impact of molecular biology on neuroscience. *Phil. Trans. Roy. Soc., B*, 354, 2021–2025.
- Crick**, F. e Koch, C. (2003). A framework for consciousness. *Nature Neuroscience*, 6, 119.
- Crisan**, D. e Doucet, A. (2002). A survey of convergence results on particle filtering methods for practitioners. *IEEE Trans. Signal Processing*, 50, 736–746.
- Cristianini**, N. e Hahn, M. (2007). *Introduction to Computational Genomics: A Case Studies Approach*. Cambridge University Press.
- Cristianini**, N. e Schölkopf, B. (2002). Support vector machines and kernel methods: The new generation of learning machines. *AIMag*, 23, 31–41.
- Cristianini**, N. e Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press.
- Crockett**, L. (1994). *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence*. Ablex.
- Croft**, W. B., Metzler, D. e Strohman, T. (2010). *Search Engines: Information Retrieval in Practice*. Addison-Wesley.
- Cross**, S. E. e Walker, E. (1994). DART: Applying knowledge based planning and scheduling to crisis action planning. In Zweben, M. e Fox, M. S. (a cura di), *Intelligent Scheduling*. Morgan Kaufmann.
- Cruse**, A. (2011). *Meaning in Language: An Introduction to Semantics and Pragmatics*. Oxford University Press.
- Culberson**, J. e Schaeffer, J. (1996). Searching with pattern databases. In *Advances in Artificial Intelligence (Lecture Notes in Artificial Intelligence 1081)*. Springer-Verlag.
- Culberson**, J. e Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, 14, 318–334.
- Cummins**, D. e Allen, C. (1998). *The Evolution of Mind*. Oxford University Press.
- Cushing**, W., Kambhampati, S., Mausam e Weld, D. S. (2007). When is temporal planning *really* temporal? In *IJCAI-07*.
- Cusumano-Towner**, M. F., Saad, F., Lew, A. K. e Mansinghka, V. K. (2019). Gen: A general-purpose probabilistic programming system with programmable inference. In *PLDI-19*.
- Cybenko**, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University.
- Cybenko**, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems*, 2, 303–314.
- Cyert**, R. e de Groot, M. (1979). Adaptive utility. In Allais, M. e Hagen, O. (a cura di), *Expected Utility Hypothesis and the Allais Paradox*. D. Reidel.
- Dagan**, I., Glickman, O. e Magnini, B. (2005). The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*.
- Daganzo**, C. (1979). *Multinomial Probit: The Theory and Its Application to Demand Forecasting*. Academic Press.
- Dagum**, P. e Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *AIJ*, 60, 141–153.
- Dagum**, P. e Luby, M. (1997). An optimal approximation algorithm for Bayesian inference. *AIJ*, 93, 1–27.
- Dai**, A. M. e Le, Q. V. (2016). Semi-supervised sequence learning. In *NeurIPS 28*.
- Dalal**, N. e Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR-05*.
- Dalvi**, N. N., Ré, C. e Suciu, D. (2009). Probabilistic databases. *CACM*, 52, 86–94.
- Daly**, R., Shen, Q. e Aitken, S. (2011). Learning Bayesian networks: Approaches and issues. *Knowledge Engineering Review*, 26, 99–157.
- Damasio**, A. R. (1999). *The Feeling of What Happens: Body and Emotion in the Making of Consciousness*. Houghton Mifflin.
- Danaher**, J. e McArthur, N. (2017). *Robot Sex: Social and Ethical Implications*. MIT Press.
- Dantzig**, G. B. (1949). Programming of interdependent activities: II. Mathematical model. *Econometrica*, 17, 200–211.
- Darwiche**, A. (2001). Recursive conditioning. *AIJ*, 126, 5–41.
- Darwiche**, A. e Ginsberg, M. L. (1992). A symbolic generalization of probability theory. In *AAAI-92*.
- Darwiche**, A. (2009). *Modeling and reasoning with Bayesian networks*. Cambridge University Press.
- Darwin**, C. (1859). *On The Origin of Species by Means of Natural Selection*. J. Murray.

- Dasgupta**, P., Chakrabarti, P. P. e de Sarkar, S. C. (1994). Agent searching in a tree and the optimality of iterative deepening. *AIJ*, 71, 195–208.
- Dasgupta**, P. e Maskin, E. (2008). On the robustness of majority rule. *Journal of the European Economic Association*, 6, 949–973.
- Dauphin**, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. e Bengio, Y. (2015). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NeurIPS* 27.
- Davidson**, D. (1980). *Essays on Actions and Events*. Oxford University Press.
- Davidson**, D. (1986). A nice derangement of epitaphs. *Philosophical Grounds of Rationality*, 4, 157–174.
- Davis**, E. (1986). *Representing and Acquiring Geographic Knowledge*. Pitman and Morgan Kaufmann.
- Davis**, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann.
- Davis**, E. (2005). Knowledge and communication: A first-order theory. *AIJ*, 166, 81–140.
- Davis**, E. (2006). The expressivity of quantifying over regions. *J. Logic and Computation*, 16, 891–916.
- Davis**, E. (2007). Physical reasoning. In van Harmelan, F., Lifschitz, V. e Porter, B. (a cura di), *The Handbook of Knowledge Representation*. Elsevier.
- Davis**, E. (2008). Pouring liquids: A study in commonsense physical reasoning. *AIJ*, 172.
- Davis**, E. (2017). Logical formalizations of commonsense reasoning: A survey. *JAIR*, 59, 651–723.
- Davis**, E. e Morgenstern, L. (2004). Introduction: Progress in formal commonsense reasoning. *AIJ*, 153, 1–12.
- Davis**, E. e Morgenstern, L. (2005). A first-order theory of communication and multi-agent plans. *J. Logic and Computation*, 15, 701–749.
- Davis**, M. (1957). A computer program for Presburger's algorithm. In *Proving Theorems (as Done by Man, Logician, or Machine)*. Proc. Summer Institute for Symbolic Logic. Seconda edizione; data di pubblicazione 1960.
- Davis**, M., Logemann, G. e Loveland, D. (1962). A machine program for theorem-proving. *CACM*, 5, 394–397.
- Davis**, M. e Putnam, H. (1960). A computing procedure for quantification theory. *JACM*, 7, 201–215.
- Dayan**, P. (1992). The convergence of TD(λ) for general λ . *Machine Learning*, 8, 341–362.
- Dayan**, P. e Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press.
- Dayan**, P. e Hinton, G. E. (1993). Feudal reinforcement learning. In *NeurIPS* 5.
- Dayan**, P. e Niv, Y. (2008). Reinforcement learning and the brain: The good, the bad and the ugly. *Current Opinion in Neurobiology*, 18, 185–196.
- de Condorcet**, M. (1785). *Essay on the Application of Analysis to the Probability of Majority Decisions*. Imprimerie Royale.
- de Dombal**, F. T., Leaper, D. J., Horrocks, J. C. e Staniland, J. R. (1974). Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, 1, 376–380.
- de Dombal**, F. T., Staniland, J. R. e Clamp, S. E. (1981). Geographical variation in disease presentation. *Medical Decision Making*, 1, 59–69.
- de Farias**, D. P. e Roy, B. V. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, 51, 839–1016.
- de Finetti**, B. (1937). La prévision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré*, 7, 1–68.
- de Finetti**, B. (1993). On the subjective meaning of probability. In Monari, P. e Cocchi, D. (a cura di), *Probabilità e Induzione*. Clueb.
- de Freitas**, J. F. G., Niranjani, M. e Gee, A. H. (2000). Sequential Monte Carlo methods to train neural network models. *Neural Computation*, 12, 933–953.
- de Ghellinck**, G. (1960). Les problèmes de décisions séquentielles. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 2, 161–179.
- de Kleer**, J. (1975). Qualitative and quantitative knowledge in classical mechanics. Tech. rep., MIT Artificial Intelligence Laboratory.
- de Kleer**, J. (1989). A comparison of ATMS and CSP techniques. In *IJCAI-89*.
- de Kleer**, J. e Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. e Moore, R. C. (a cura di), *Formal Theories of the Commonsense World*. Ablex.
- de Marcken**, C. (1996). *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.
- De Morgan**, A. (1864). On the syllogism, No. IV e on the logic of relations. *Transaction of the Cambridge Philosophical Society*, X, 331–358.
- de Salvo Braz**, R., Amir, E. e Roth, D. (2007). Lifted first-order probabilistic inference. In Getoor, L. e Taskar, B. (a cura di), *Introduction to Statistical Relational Learning*. MIT Press.
- Deacon**, T. W. (1997). *The Symbolic Species: The Coevolution of Language and the Brain*. W. W. Norton.

- Deale**, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G. e Daun, B. (1994). The space shuttle ground processing scheduling system. In Zweben, M. e Fox, M. (a cura di), *Intelligent Scheduling*. Morgan Kaufmann.
- Dean**, J., Patterson, D. A. e Young, C. (2018). A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 38, 21–29.
- Dean**, T., Basye, K., Chekaluk, R. e Hyun, S. (1990). Coping with uncertainty in a control system for navigation and exploration. In *AAAI-90*.
- Dean**, T. e Boddy, M. (1988). An analysis of time-dependent planning. In *AAAI-88*.
- Dean**, T., Firby, R. J. e Miller, D. (1990). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 6, 381–398.
- Dean**, T., Kaelbling, L. P., Kirman, J. e Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *AAAI-93*.
- Dean**, T. e Kanazawa, K. (1989a). A model for projection and action. In *IJCAI-89*.
- Dean**, T. e Kanazawa, K. (1989b). A model for reasoning about persistence and causation. *Computational Intelligence*, 5, 142–150.
- Dean**, T. e Wellman, M. P. (1991). *Planning and Control*. Morgan Kaufmann.
- Dearden**, R., Friedman, N. e Andre, D. (1999). Model-based Bayesian exploration. In *UAI-99*.
- Dearden**, R., Friedman, N. e Russell, S. J. (1998). Bayesian Q-learning. In *AAAI-98*.
- Debevec**, P., Taylor, C. e Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proc. 23rd Annual Conference on Computer Graphics (SIGGRAPH)*.
- Debreu**, G. (1960). Topological methods in cardinal utility theory. In Arrow, K. J., Karlin, S. e Suppes, P. (a cura di), *Mathematical Methods in the Social Sciences, 1959*. Stanford University Press.
- Dechter**, A. e Dechter, R. (1987). Removing redundancies in constraint networks. In *AAAI-87*.
- Dechter**, R. (1990a). Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *AIJ*, 41, 273–312.
- Dechter**, R. (1990b). On the expressiveness of networks with hidden variables. In *AAAI-90*.
- Dechter**, R. (1999). Bucket elimination: A unifying framework for reasoning. *AIJ*, 113, 41–85.
- Dechter**, R. e Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *JACM*, 32, 505–536.
- Dechter**, R. e Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *AIJ*, 34, 1–38.
- Dechter**, R. e Pearl, J. (1989). Tree clustering for constraint networks. *AIJ*, 38, 353–366.
- Dechter**, R. e Rish, I. (2003). Mini-buckets: A general scheme for bounded inference. *JACM*, 50, 107–153.
- Dechter**, R. (2003). *Constraint Processing*. Morgan Kaufmann.
- Dechter**, R. (2019). *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms* (2nd edition). Morgan & Claypool.
- Dechter**, R. e Frost, D. (2002). Backjump-based backtracking for constraint satisfaction problems. *AIJ*, 136, 147–188.
- Dechter**, R. e Mateescu, R. (2007). AND/OR search spaces for graphical models. *AIJ*, 171, 73–106.
- DeCoste**, D. e Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, 46, 161–190.
- Dedekind**, R. (1888). *Was sind und was sollen die Zahlen*. Braunschweig, Germany.
- Deerwester**, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W. e Harshman, R. A. (1990). Indexing by latent semantic analysis. *J. American Society for Information Science*, 41, 391–407.
- DeGroot**, M. H. (1970). *Optimal Statistical Decisions*. McGraw-Hill.
- DeGroot**, M. H. e Schervish, M. J. (2001). *Probability and Statistics* (3rd edition). Addison Wesley.
- Dehaene**, S. (2014). *Consciousness and the Brain: Deciphering How the Brain Codes Our Thoughts*. Penguin Books.
- Del Moral**, P., Doucet, A. e Jasra, A. (2006). Sequential Monte Carlo samplers. *J. Royal Statistical Society*, 68, 411–436.
- Del Moral**, P. (2004). *Feynman-Kac Formulae, Genealogical and Interacting Particle Systems with Applications*. Springer-Verlag.
- Delgrande**, J. e Schaub, T. (2003). On the relation between Reiter's default logic and its (major) variants. In *Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*.
- Delling**, D., Sanders, P., Schulze, D. e Wagner, D. (2009). Engineering route planning algorithms. In Lerner, J., Wagner, D. e Zweig, K. (a cura di), *Algorithmics, LNCS*. Springer-Verlag.
- Dempster**, A. P. (1968). A generalization of Bayesian inference. *J. Royal Statistical Society*, 30 (Series B), 205–247.
- Dempster**, A. P., Laird, N. e Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society*, 39 (Series B), 1–38.

- Denardo, E. V.** (1967). Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9, 165–177.
- Deng, J.**, Dong, W., Socher, R., Li, L.-J., Li, K. e Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR-09*.
- Deng, L.** (2016). Deep learning: From speech recognition to language and multimodal processing. *APSIPA Transactions on Signal and Information Processing*, 5.
- Deng, L.**, Yu, D., et al. (2014). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7, 197–387.
- Deng, X.** e Papadimitriou, C. H. (1990). Exploring an unknown graph. In *FOCS-90*.
- Deng, X.** e Papadimitriou, C. H. (1994). On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19, 257–266.
- Denney, E.**, Fischer, B. e Schumann, J. (2006). An empirical evaluation of automated theorem provers in software certification. *Int. J. AI Tools*, 15, 81–107.
- D'Épenoux, F.** (1963). A probabilistic production and inventory problem. *A probabilistic production and inventory problem*, 10, 98–108.
- Dervovic, D.**, Herbster, M., Mountney, P., Severini, S., Usher, N. e Wossnig, L. (2018). Quantum linear systems algorithms: A primer. arXiv:1802.08227.
- Descartes, R.** (1637). Discourse on method. In Cottingham, J., Stoothoff, R. e Murdoch, D. (a cura di), *The Philosophical Writings of Descartes*, Vol. I. Cambridge University Press, Cambridge.
- Descotte, Y.** e Latombe, J.-C. (1985). Making compromises among antagonist constraints in a planner. *AIJ*, 27, 183–217.
- Deshpande, I.**, Hu, Y.-T., Sun, R., Pyrros, A., Siddiqui, N., Koyejo, S., Zhao, Z., Forsyth, D. e Schwing, A. (2019). Max-sliced Wasserstein distance and its use for GANs. In *CVPR-19*.
- Deutscher, G.** (2010). *Through the Language Glass: Why the World Looks Different in Other Languages*. Metropolitan Books.
- Devlin, J.**, Chang, M.-W., Lee, K. e Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805.
- Devlin, K.** (2018). *Turned On: Science, Sex and Robots*. Bloomsbury.
- Devroye, L.** (1987). *A course in density estimation*. Birkhauser.
- Dias, M. B.**, Zlot, R., Kalra, N. e Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proc. IEEE*, 94, 1257–1270.
- Dickmanns, E. D.** e Zapp, A. (1987). Autonomous high speed road vehicle guidance by computer vision. In *Automatic Control—World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*.
- Dietterich, T.** (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*, 13, 227–303.
- Dijkstra, E. W.** (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271.
- Dijkstra, E. W.** (1984). The threats to computing science. In *ACM South Central Regional Conference*.
- Ding, Y.**, Sohn, J. H., Kawczynski, M. G., Trivedi, H., Harnish, R., Jenkins, N. W., Litviev, D., Copeland, T. P., Aboian, M. S., Mari Aparici, C., et al. (2018). A deep learning model to predict a diagnosis of alzheimer disease by using 18F-FDG PET of the brain. *Radiology*, p. 180958.
- Dinh, H.**, Russell, A. e Su, Y. (2007). On the value of good advice: The complexity of A* with accurate heuristics. In *AAAI-07*.
- Dissanayake, G.**, Newman, P., Clark, S., Durrant-Whyte, H. e Csorba, M. (2001). A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17, 229–241.
- Dittmer, S.** e Jensen, F. (1997). Myopic value of information in influence diagrams. In *UAI-97*.
- Do, M.** e Kambhampati, S. (2003). Planning as constraint satisfaction: solving the planning graph by compiling it into CSP. *AIJ*, 132, 151–182.
- Do, M. B.** e Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In *ECP-01*.
- Doctorow, C.** (2001). Metacrap: Putting the torch to seven straw-men of the meta-utopia. www.well.com/doctorow/metacrap.htm.
- Doctorow, C.** e Stross, C. (2012). *The Rapture of the Nerds: A Tale of the Singularity, Posthumanity, and Awkward Social Situations*. Tor Books.
- Dodd, L.** (1988). The inside/outside algorithm: Grammatical inference applied to stochastic context-free grammars. Tech. rep., Royal Signals and Radar Establishment, Malvern.

- Domingos**, P. e Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero–one loss. *Machine Learning*, 29, 103–30.
- Domingos**, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10), 78–87.
- Domingos**, P. (2015). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books.
- Dong**, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S. e Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD-14*.
- Doorenbos**, R. (1994). Combining left and right unlinking for matching a large number of learned rules. In *AAAI-94*.
- Doran**, J. e Michie, D. (1966). Experiments with the graph traverser program. *Proc. Roy. Soc.*, 294, Series A, 235–259.
- Dorf**, R. C. e Bishop, R. H. (2004). *Modern Control Systems* (10th edition). Prentice-Hall.
- Dorigo**, M., Birattari, M., Blum, C., Clerc, M., Stützle, T. e Winfield, A. (2008). *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22–24, 2008, Proceedings*, Vol. 5217. Springer-Verlag.
- Doshi-Velez**, F. e Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv:1702.08608.
- Doucet**, A. (1997). *Monte Carlo methods for Bayesian estimation of hidden Markov models: Application to radiation signals*. Ph.D. thesis, Université de Paris-Sud.
- Doucet**, A., de Freitas, J. F. G. e Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.
- Doucet**, A., de Freitas, J. F. G., Murphy, K. e Russell, S. J. (2000). Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *UAI-00*.
- Doucet**, A. e Johansen, A. M. (2011). A tutorial on particle filtering and smoothing: Fifteen years later. In Crisan, D. e Rozovskii, B. (a cura di), *Oxford Handbook of Nonlinear Filtering*. Oxford.
- Dowty**, D., Wall, R. e Peters, S. (1991). *Introduction to Montague Semantics*. D. Reidel.
- Doyle**, J. (1979). A truth maintenance system. *AIJ*, 12, 231–272.
- Doyle**, J. (1983). What is rational psychology? Toward a modern mental philosophy. *AIMag*, 4, 50–53.
- Drabble**, B. (1990). Mission scheduling for spacecraft: Diaries of T-SCHED. In *Expert Planning Systems*. Institute of Electrical Engineers.
- Dragan**, A. D., Lee, K. C. e Srinivasa, S. (2013). Legibility and predictability of robot motion. In *HRI-13*.
- Dredze**, M., Crammer, K. e Pereira, F. (2008). Confidence-weighted linear classification. In *ICML-08*.
- Dressel**, J. e Farid, H. (2018). The accuracy, fairness, and limits of predicting recidivism. *Science Advances*, 4, eaao5580.
- Dreyfus**, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason*. Harper and Row.
- Dreyfus**, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press.
- Dreyfus**, H. L. e Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Blackwell.
- Dreyfus**, S. E. (1962). The numerical solution of variational problems. *J. Math. Anal. e Appl.*, 5, 30–45.
- Dreyfus**, S. E. (1969). An appraisal of some shortest-paths algorithms. *Operations Research*, 17, 395–412.
- Dreyfus**, S. E. (1990). Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure. *J. Guidance, Control, and Dynamics*, 13, 926–928.
- Du**, S. S., Lee, J. D., Li, H., Wang, L. e Zhai, X. (2018). Gradient descent finds global minima of deep neural networks. arXiv:1811.03804.
- Dubois**, D. e Prade, H. (1994). A survey of belief revision and updating rules in various uncertainty models. *Int. J. Intelligent Systems*, 9, 61–100.
- Duda**, R. O. e Hart, P. E. (1973). *Pattern classification and scene analysis*. Wiley.
- Duda**, R. O., Hart, P. E. e Stork, D. G. (2001). *Pattern Classification* (2nd edition). Wiley.
- Dudek**, G. e Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press.
- Duffy**, D. (1991). *Principles of Automated Theorem Proving*. John Wiley & Sons.
- Dunn**, H. L. (1946). Record linkage. *Am. J. Public Health*, 36, 1412–1416.
- Durfee**, E. H. e Lesser, V. R. (1989). Negotiating task decomposition and allocation using partial global planning. In Huhns, M. e Gasser, L. (a cura di), *Distributed AI*, Vol. 2. Morgan Kaufmann.
- Durme**, B. V. e Pasca, M. (2008). Finding cars, goddesses and enzymes: Parametrizable acquisition of labeled instances for open-domain information extraction. In *AAAI-08*.
- Dwork**, C. (2008). Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*.

- Dwork**, C., Hardt, M., Pitassi, T., Reingold, O. e Zemel, R. (2012). Fairness through awareness. In *Proc. 3rd innovations in theoretical computer science conference*.
- Dwork**, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9, 211–407.
- Dyson**, F. (2004). A meeting with Enrico Fermi. *Nature*, 427, 297.
- Dyson**, G. (1998). *Darwin among the machines : the evolution of global intelligence*. Perseus Books.
- Earley**, J. (1970). An efficient context-free parsing algorithm. *CACM*, 13, 94–102.
- Ebendt**, R. e Drechsler, R. (2009). Weighted A* search—unifying view and application. *AIJ*, 173, 1310–1342.
- Eckerle**, J., Chen, J., Sturtevant, N. R., Zilles, S. e Holte, R. C. (2017). Sufficient conditions for node expansion in bidirectional heuristic search. In *ICAPS-17*.
- Eckhouse**, L., Lum, K., Conti-Cook, C. e Ciccolini, J. (2019). Layers of bias: A unified approach for understanding problems with risk assessment. *Criminal Justice and Behavior*, 46, 185–209.
- Edelkamp**, S. (2009). Scaling search with symbolic pattern databases. In *Model Checking and Artificial Intelligence (MOCHART)*.
- Edelkamp**, S. e Schrödl, S. (2012). *Heuristic Search*. Morgan Kaufmann.
- Edmonds**, J. (1965). Paths, trees, and flowers. *Canadian J. of Mathematics*, 17, 449–467.
- Edwards**, P. (a cura di). (1967). *The Encyclopedia of Philosophy*. Macmillan.
- Eiter**, T., Leone, N., Mateis, C., Pfeifer, G. e Scarcello, F. (1998). The KR system dlv: Progress report, comparisons and benchmarks. In *KR-98*.
- Elio**, R. (a cura di). (2002). *Common Sense, Reasoning, and Rationality*. Oxford University Press.
- Elkan**, C. (1997). Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.
- Ellsberg**, D. (1962). *Risk, Ambiguity, and Decision*. Ph.D. thesis, Harvard University.
- Elman**, J. L. (1990). Finding structure in time. *Cognitive Science*, 14, 179–211.
- Elman**, J. L., Bates, E., Johnson, M., Karmiloff-Smith, A., Parisi, D. e Plunkett, K. (1997). *Rethinking Innateness*. MIT Press.
- Elo**, A. E. (1978). *The rating of chess players: Past and present*. Arco Publishing.
- Elsken**, T., Metzen, J. H. e Hutter, F. (2018). Neural architecture search: A survey. arXiv:1808.05377.
- Empson**, W. (1953). *Seven Types of Ambiguity*. New Directions.
- Enderton**, H. B. (1972). *A Mathematical Introduction to Logic*. Academic Press.
- Engel**, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D. e Simonyan, K. (2017). Neural audio synthesis of musical notes with wavenet autoencoders. In *Proc. 34th International Conference on Machine Learning-Volume 70*.
- Epstein**, R., Roberts, G. e Beber, G. (a cura di). (2008). *Parsing the Turing test*. Springer.
- Erdmann**, M. A. e Mason, M. (1988). An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4, 369–379.
- Ernst**, H. A. (1961). *MH-J, a Computer-Operated Mechanical Hand*. Ph.D. thesis, MIT.
- Ernst**, M., Millstein, T. e Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *IJCAI-97*.
- Erol**, K., Hendler, J. e Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *AAAI-94*.
- Erol**, K., Hendler, J. e Nau, D. S. (1996). Complexity results for HTN planning. *AIJ*, 18, 69–93.
- Erol**, Y., Li, L., Ramsundar, B. e Russell, S. J. (2013). The extended parameter filter. In *ICML-13*.
- Erol**, Y., Wu, Y., Li, L. e Russell, S. J. (2017). A nearly-black-box online algorithm for joint parameter and state estimation in temporal models. In *AAAI-17*.
- Esteva**, A., Kuprel, B., Novoa, R. A., Ko, J., Swet-ter, S. M., Blau, H. M. e Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542, 115.
- Etzioni**, A. (2004). *From Empire to Community: A New Approach to International Relation*. Palgrave Macmillan.
- Etzioni**, A. e Etzioni, O. (2017a). Incorporating ethics into artificial intelligence. *The Journal of Ethics*, 21, 403–418.
- Etzioni**, A. e Etzioni, O. (2017b). Should artificial intelligence be regulated? *Issues in Science and Technology*, Summer.
- Etzioni**, O. (1989). Tractable decision-analytic control. In *Proc. First International Conference on Knowledge Representation and Reasoning*.
- Etzioni**, O., Banko, M., Soderland, S. e Weld, D. S. (2008). Open information extraction from the web. *CACM*, 51.
- Etzioni**, O., Hanks, S., Weld, D. S., Draper, D., Lesh, N. e Williamson, M. (1992). An approach to planning with incomplete information. In *KR-92*.
- Etzioni**, O., Banko, M. e Cafarella, M. J. (2006). Machine reading. In *AAAI-06*.

- Etzioni**, O., Cafarella, M. J., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S. e Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *AIJ*, 165(1), 91–134.
- Evans**, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky, M. L. (a cura di), *Semantic Information Processing*. MIT Press.
- Fagin**, R., Halpern, J. Y., Moses, Y. e Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press.
- Fahlman**, S. E. (1974). A planning system for robot construction tasks. *AIJ*, 5, 1–49.
- Faugeras**, O. (1992). What can be seen in three dimensions with an uncalibrated stereo rig? In *ECCV*, Vol. 588 of *Lecture Notes in Computer Science*.
- Faugeras**, O., Luong, Q.-T. e Papadopoulo, T. (2001). *The Geometry of Multiple Images*. MIT Press.
- Fawcett**, T. e Provost, F. (1997). Adaptive fraud detection. *Data mining and knowledge discovery*, 1, 291–316.
- Fearing**, R. S. e Hollerbach, J. M. (1985). Basic solid mechanics for tactile sensing. *Int. J. Robotics Research*, 4, 40–54.
- Featherstone**, R. (1987). *Robot Dynamics Algorithms*. Kluwer Academic Publishers.
- Feigenbaum**, E. A. (1961). The simulation of verbal learning behavior. *Proc. Western Joint Computer Conference*, 19, 121–131.
- Feigenbaum**, E. A., Buchanan, B. G. e Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In Meltzer, B. e Michie, D. (a cura di), *Machine Intelligence 6*. Edinburgh University Press.
- Feldman**, J. e Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.
- Feldman**, J. e Yakimovsky, Y. (1974). Decision theory and artificial intelligence I: Semantics-based region analyzer. *AIJ*, 5, 349–371.
- Feldman**, M. (2017). Oak Ridge readies Summit supercomputer for 2018 debut. *Top500.org*, bit.ly/2ERRFr9.
- Fellbaum**, C. (2001). *Wordnet: An Electronic Lexical Database*. MIT Press.
- Fellegi**, I. e Sunter, A. (1969). A theory for record linkage. *JASA*, 64, 1183–1210.
- Felner**, A., Korf, R. E. e Hanan, S. (2004). Additive pattern database heuristics. *JAIR*, 22, 279–318.
- Felner**, A. (2018). Position paper: Using early goal test in A*. In *Eleventh Annual Symposium on Combinatorial Search*.
- Felner**, A., Korf, R. E., Meshulam, R. e Holte, R. C. (2007). Compressed pattern databases. *JAIR*, 30.
- Felner**, A., Zahavi, U., Holte, R. C., Schaeffer, J., Sturtevant, N. R. e Zhang, Z. (2011). Inconsistent heuristics in theory and practice. *AIJ*, 175, 1570–1603.
- Felzenszwab**, P. e McAllester, D. A. (2007). The generalized A* architecture. *JAIR*.
- Fenton**, N. e Neil, M. (2018). *Risk Assessment and Decision Analysis with Bayesian Networks* (2nd edition). Chapman and Hall.
- Ferguson**, T. (1992). Mate with knight and bishop in kriegspiel. *Theoretical Computer Science*, 96, 389–403.
- Ferguson**, T. (1995). Mate with the two bishops in kriegspiel. www.math.ucla.edu/~tom/papers.
- Ferguson**, T. (2001). *Optimal Stopping and Applications*. www.math.ucla.edu/~tom/Stopping/Contents.html.
- Ferguson**, T. (1973). Bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1, 209–230.
- Fern**, A., Natarajan, S., Judah, K. e Tadepalli, P. (2014). A decision-theoretic model of assistance. *JAIR*, 50, 71–104.
- Fernandez**, J. M. F. e Mahlmann, T. (2018). The Dota 2 bot competition. *IEEE Transactions on Games*.
- Ferraris**, P. e Giunchiglia, E. (2000). Planning as satisfiability in nondeterministic domains. In *AAAI-00*.
- Ferriss**, T. (2007). *The 4-Hour Workweek*. Crown.
- Ferrucci**, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefler, N. e Welty, C. (2010). Building Watson: An overview of the DeepQA project. *AI Magazine*, Fall.
- Fikes**, R. E., Hart, P. E. e Nilsson, N. J. (1972). Learning and executing generalized robot plans. *AIJ*, 3, 251–288.
- Fikes**, R. E. e Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ*, 2, 189–208.
- Fikes**, R. E. e Nilsson, N. J. (1993). STRIPS, a retrospective. *AIJ*, 59, 227–232.
- Fine**, S., Singer, Y. e Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32.
- Finn**, C., Abbeel, P. e Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. 34th International Conference on Machine Learning-Volume 70*.
- Finney**, D. J. (1947). *Probit analysis: A statistical treatment of the sigmoid response curve*. Cambridge University Press.
- Firoiu**, V., Whitney, W. F. e Tenenbaum, J. B. (2017). Beating the world's best at Super Smash Bros. with deep reinforcement learning. *arXiv:1702.06230*.

- Firth, J.** (1957). *Papers in Linguistics*. Oxford University Press.
- Fisher, R. A.** (1922). On the mathematical foundations of theoretical statistics. *Phil. Trans. Roy. Soc., A*, 222, 309–368.
- Fix, E. e Hodges, J. L.** (1951). Discriminatory analysis—Nonparametric discrimination: Consistency properties. Tech. rep., USAF School of Aviation Medicine.
- Floreano, D., Zufferey, J. C., Srinivasan, M. V. e Ellington, C.** (2009). *Flying Insects and Robots*. Springer.
- Floyd, R. W.** (1962). Algorithm 97: Shortest path. *CACM*, 5, 345.
- Fogel, D. B.** (2000). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.
- Fogel, L. J., Owens, A. J. e Walsh, M. J.** (1966). *Artificial Intelligence through Simulated Evolution*. Wiley.
- Forbes, J., Huang, T., Kanazawa, K. e Russell, S. J.** (1995). The BATmobile: Towards a Bayesian automated taxi. In *IJCAI-95*.
- Forbus, K. D.** (1985). Qualitative process theory. In Bobrow, D. (a cura di), *Qualitative Reasoning About Physical Systems*. MIT Press.
- Forbus, K. D. e de Kleer, J.** (1993). *Building Problem Solvers*. MIT Press.
- Forbus, K. D., Hinrichs, T. R., De Kleer, J. e Usher, M.** (2010). FIRE: Infrastructure for experience-based systems with common sense. In *AAAI Fall Symposium: Commonsense Knowledge*.
- Ford, K. M. e Hayes, P. J.** (1995). Turing Test considered harmful. In *IJCAI-95*.
- Ford, L. R.** (1956). Network flow theory. Tech. rep., RAND Corporation.
- Ford, M.** (2015). *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books.
- Ford, M.** (2018). *Architects of Intelligence*. Packt.
- Forestier, J.-P. e Varaiya, P.** (1978). Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control*, 23, 298–304.
- Forgy, C.** (1981). OPS5 user's manual. Technical report, Computer Science Department, Carnegie-Mellon University.
- Forgy, C.** (1982). A fast algorithm for the many patterns/many objects match problem. *AIJ*, 19, 17–37.
- Forster, E. M.** (1909). *The Machine Stops*. Sheba Blake.
- Forsyth, D. e Ponce, J.** (2002). *Computer Vision: A Modern Approach*. Prentice Hall.
- Fouhey, D., Kuo, W.-C., Efros, A. e Malik, J.** (2018). From lifestyle vlogs to everyday interactions. In *CVPR-18*.
- Fourier, J.** (1827). Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824; partie mathématique. *Histoire de l'Académie Royale des Sciences de France*, 7, xlvi–lv.
- Fox, C. e Tversky, A.** (1995). Ambiguity aversion and comparative ignorance. *Quarterly Journal of Economics*, 110, 585–603.
- Fox, D., Burgard, W., Dellaert, F. e Thrun, S.** (1999). Monte Carlo localization: Efficient position estimation for mobile robots. In *AAAI-99*.
- Fox, M. S.** (1990). Constraint-guided scheduling: A short history of research at CMU. *Computers in Industry*, 14, 79–88.
- Fox, M. S., Allen, B. e Strohm, G.** (1982). Job shop scheduling: An investigation in constraint-directed reasoning. In *AAAI-82*.
- Franco, J. e Paull, M.** (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5, 77–87.
- Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. e Pineau, J.** (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11, 219–354.
- Frank, I., Basin, D. A. e Matsubara, H.** (1998). Finding optimal strategies for imperfect information games. In *AAAI-98*.
- Frank, R. H. e Cook, P. J.** (1996). *The Winner-Take-All Society*. Penguin.
- Frans, K., Ho, J., Chen, X., Abbeel, P. e Schulman, J.** (2018). Meta learning shared hierarchies. In *ICLR-18*.
- Franz, A. e Brants, T.** (2006). All our n-gram are belong to you. Google blog, ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html.
- Frege, G.** (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, Berlin. Traduzione inglese in van Heijenoort (1967).
- Freitag, D. e McCallum, A.** (2000). Information extraction with hmm structures learned by stochastic optimization. In *AAAI-00*.
- Freuder, E. C.** (1978). Synthesizing constraint expressions. *CACM*, 21, 958–966.
- Freuder, E. C.** (1982). A sufficient condition for backtrack-free search. *JACM*, 29, 24–32.
- Freuder, E. C.** (1985). A sufficient condition for backtrack-bounded search. *JACM*, 32, 755–761.
- Freund, Y. e Schapire, R. E.** (1996). Experiments with a new boosting algorithm. In *ICML-96*.

- Freund**, Y. e Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37, 277–296.
- Frey**, B. J. (1998). *Graphical models for machine learning and digital communication*. MIT Press.
- Frey**, C. B. e Osborne, M. A. (2017). The future of employment: How susceptible are jobs to computerisation? *Technological forecasting and social change*, 114, 254–280.
- Friedberg**, R. M. (1958). A learning machine: Part I. *IBM Journal of Research and Development*, 2, 2–13.
- Friedberg**, R. M., Dunham, B. e North, T. (1959). A learning machine: Part II. *IBM Journal of Research and Development*, 3, 282–287.
- Friedman**, G. J. (1959). Digital simulation of an evolutionary process. *General Systems Yearbook*, 4, 171–184.
- Friedman**, J., Hastie, T. e Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28, 337–374.
- Friedman**, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 29, 1189–1232.
- Friedman**, N. (1998). The Bayesian structural EM algorithm. In *UAI-98*.
- Friedman**, N. e Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *UAI-96*.
- Friedman**, N. e Koller, D. (2003). Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50, 95–125.
- Friedman**, N., Murphy, K. e Russell, S. J. (1998). Learning the structure of dynamic probabilistic networks. In *UAI-98*.
- Friedman**, N. (2004). Inferring cellular networks using probabilistic graphical models. *Science*, 303.
- Fruhwirth**, T. e Abdennadher, S. (2003). *Essentials of constraint programming*. Cambridge University Press.
- Fuchs**, J. J., Gasquet, A., Olalainy, B. e Currie, W. (1990). PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*. Institute of Electrical Engineers.
- Fudenberg**, D. e Tirole, J. (1991). *Game theory*. MIT Press.
- Fukunaga**, A. S., Rabideau, G., Chien, S. e Yan, D. (1997). ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proc. International Symposium on AI, Robotics and Automation in Space*.
- Fukushima**, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36, 193–202.
- Fukushima**, K. e Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*. Springer.
- Fuller**, S. B., Straw, A. D., Peek, M. Y., Murray, R. M. e Dickinson, M. H. (2014). Flying Drosophila stabilize their vision-based velocity controller by sensing wind with their antennae. *Proc. National Academy of Sciences of the United States of America*, 111 13, E1182–91.
- Fung**, C., Yoon, C. J. M. e Beschastnikh, I. (2018). Mitigating sybils in federated learning poisoning. arXiv:1808.04866.
- Fung**, R. e Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *UAI 5*.
- Gaddum**, J. H. (1933). Reports on biological standard III: Methods of biological assay depending on a quantal response. Special report series of the medical research council, Medical Research Council.
- Gaifman**, H. (1964a). Concerning measures in first order calculi. *Israel J. Mathematics*, 2, 1–18.
- Gaifman**, H. (1964b). Concerning measures on Boolean algebras. *Pacific J. Mathematics*, 14, 61–73.
- Gallaire**, H. e Minker, J. (a cura di). (1978). *Logic and Databases*. Plenum.
- Gallier**, J. H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row.
- Galton**, F. (1886). Regression towards mediocrity in hereditary stature. *J. Anthropological Institute of Great Britain and Ireland*, 15, 246–263.
- Gamba**, A., Gamberini, L., Palmieri, G. e Sanna, R. (1961). Further experiments with PAPA. *Nuovo Cimento Supplemento*, 20, 221–231.
- Gandomi**, A. e Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management*, 35, 137–144.
- Gao**, J. (2014). Machine learning applications for data center optimization. Google Research.
- García**, J. e Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *JMLR*, 16, 1437–1480.
- Gardner**, M. (1968). *Logic Machines, Diagrams and Boolean Algebra*. Dover.
- Garey**, M. R. e Johnson, D. S. (1979). *Computers and Intractability*. W. H. Freeman.

- Gaschnig, J.** (1977). A general backtrack algorithm that eliminates most redundant tests. In *IJCAI-77*.
- Gaschnig, J.** (1979). Performance measurement and analysis of certain search algorithms. Technical report, Computer Science Department, Carnegie-Mellon University.
- Gasser, R.** (1995). *Efficiently harnessing computational resources for exhaustive search*. Ph.D. thesis, ETH Zürich.
- Gat, E.** (1998). Three-layered architectures. In Kortenkamp, D., Bonasso, R. P. e Murphy, R. (a cura di), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.
- Gatys, L. A.**, Ecker, A. S. e Bethge, M. (2016). Image style transfer using convolutional neural networks. In *CVPR-16*.
- Gauci, J.**, Conti, E., Liang, Y., Virochsiri, K., He, Y., Kaden, Z., Narayanan, V. e Ye, X. (2018). Horizon: la piattaforma open source di Facebook per l'apprendimento con rinforzo. arXiv:1811.00260.
- Gauss, C. F.** (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Sumtibus F. Perthes et I. H. Besser, Hamburg.
- Gauss, C. F.** (1829). Beiträge zur theorie der algebraischen gleichungen. *Werke*, 3, 71–102.
- Gazzaniga, M.** (2018). *The Consciousness Instinct*. Farrar, Straus and Girou.
- Gebru, T.**, Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H. M., III, H. D. e Crawford, K. (2018). Datasheets for datasets. arXiv:1803.09010.
- Geiger, D.**, Verma, T. e Pearl, J. (1990a). d-separation: From theorems to algorithms. In Henrion, M., Shachter, R. D., Kanal, L. N. e Lemmer, J. F. (a cura di), *UAI-90*. Elsevier.
- Geiger, D.**, Verma, T. e Pearl, J. (1990b). Identifying independence in Bayesian networks. *Networks*, 20, 507–534.
- Gelb, A.** (1974). *Applied Optimal Estimation*. MIT Press.
- Gelernter, H.** (1959). Realization of a geometry-theorem proving machine. In *Proc. an International Conference on Information Processing*. UNESCO House.
- Gelfond, M.** e Lifschitz, V. (1988). Compiling circumscriptive theories into logic programs. In *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*.
- Gelfond, M.** (2008). Answer sets. In van Harmelen, F., Lifschitz, V. e Porter, B. (a cura di), *Handbook of Knowledge Representation*. Elsevier.
- Gelman, A.** (2004). Exploratory data analysis for complex models. *Journal of Computational and Graphical Statistics*, 13, 755–779.
- Gelman, A.**, Carlin, J. B., Stern, H. S. e Rubin, D. (1995). *Bayesian Data Analysis*. Chapman & Hall.
- Geman, S.** e Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *PAMI*, 6, 721–741.
- Gene Ontology Consortium**, The. (2008). The gene ontology project in 2008. *Nucleic Acids Research*, 36(D440–D444).
- Genesereth, M. R.** (1984). The use of design descriptions in automated diagnosis. *AIJ*, 24, 411–436.
- Genesereth, M. R.** e Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann.
- Genesereth, M. R.** e Nourbakhsh, I. (1993). Time-saving tips for problem solving with incomplete information. In *AAAI-93*.
- Genesereth, M. R.** e Smith, D. E. (1981). Meta-level architecture. Memo, Computer Science Department, Stanford University.
- Gent, I.**, Petrie, K. e Puget, J.-F. (2006). Symmetry in constraint programming. In Rossi, F., van Beek, P. e Walsh, T. (a cura di), *Handbook of Constraint Programming*. Elsevier.
- Géron, A.** (2019). *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly.
- Gers, F. A.**, Schmidhuber, J. e Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12, 2451–2471.
- Getoor, L.** e Taskar, B. (a cura di). (2007). *Introduction to Statistical Relational Learning*. MIT Press.
- Ghaheri, A.**, Shoar, S., Naderan, M. e Hoseini, S. (2015). The applications of genetic algorithms in medicine. *Oman medical journal*, 30, 406–416.
- Ghahramani, Z.** (1998). Learning dynamic Bayesian networks. In *Adaptive Processing of Sequences and Data Structures*.
- Ghahramani, Z.** (2005). Tutorial on nonparametric Bayesian methods. Given at the UAI-05 Conference.
- Ghallab, M.**, Howe, A., Knoblock, C. A. e McDermott, D. (1998). PDDL—The planning domain definition language. Tech. rep., Yale Center for Computational Vision and Control.
- Ghalla, M.** e Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *AIPS-94*.
- Ghalla, M.**, Nau, D. S. e Traverso, P. (2004). *Automated Planning: Theory and practice*. Morgan Kaufmann.
- Ghalla, M.**, Nau, D. S. e Traverso, P. (2016). *Automated Planning and aTing*. Cambridge University Press.
- Gibbs, R. W.** (2006). Metaphor interpretation as embodied simulation. *Mind*, 21, 434–458.

- Gibson, J. J.** (1950). *The Perception of the Visual World*. Houghton Mifflin.
- Gibson, J. J.** (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin.
- Gibson, J. J., Olum, P. e Rosenblatt, F.** (1955). Parallax and perspective during aircraft landings. *American Journal of Psychology*, 68, 372–385.
- Gilks, W. R., Richardson, S. e Spiegelhalter, D. J.** (a cura di). (1996). *Markov chain Monte Carlo in practice*. Chapman and Hall.
- Gilks, W. R., Thomas, A. e Spiegelhalter, D. J.** (1994). A language and program for complex Bayesian modelling. *The Statistician*, 43, 169–178.
- Gilks, W. R. e Berzuini, C.** (2001). Following a moving target—Monte Carlo inference for dynamic Bayesian models. *J. Royal Statistical Society*, 63, 127–146.
- Gilks, W. R. e Wild, P. P.** (1992). Adaptive rejection sampling for Gibbs sampling. *Applied Statistics*, 41, 337–348.
- Gillies, D. B.** (1959). Solutions to general non-zero-sum games. In Tucker, A. W. e Luce, L. D. (a cura di), *Contributions to the Theory of Games, volume IV*. Princeton University Press.
- Gilmore, P. C.** (1960). A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4, 28–35.
- Gilpin, A., Sandholm, T. e Sorensen, T.** (2008). A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS-08*.
- Ginsberg, M. L.** (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann.
- Ginsberg, M. L.** (2001). GIB: Imperfect information in a computationally challenging game. *JAIR*, 14, 303–358.
- Gionis, A., Indyk, P. e Motwani, R.** (1999). Similarity search in high dimensions via hashing. In *Proc. 25th Very Large Database (VLDB) Conference*.
- Girshick, R., Donahue, J., Darrell, T. e Malik, J.** (2016). Region-based convolutional networks for accurate object detection and segmentation. *PAMI*, 38, 142–58.
- Gittins, J. C.** (1989). *Multi-Armed Bandit Allocation Indices*. Wiley.
- Gittins, J. C. e Jones, D. M.** (1974). A dynamic allocation index for the sequential design of experiments. In Gani, J. (a cura di), *Progress in Statistics*. North-Holland.
- Glanc, A.** (1978). On the etymology of the word “robot”. *SIGART Newsletter*, 67, 12.
- Glickman, M. E.** (1999). Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, 48, 377–394.
- Glorot, X., Bordes, A. e Bengio, Y.** (2011). Deep sparse rectifier neural networks. In *AISTATS'2011*.
- Glover, F. e Laguna, M.** (a cura di). (1997). *Tabu search*. Kluwer.
- Gluss, B.** (1959). An optimum policy for detecting a fault in a complex system. *Operations Research*, 7, 468–477.
- Godefroid, P.** (1990). Using partial orders to improve automatic verification methods. In *Proc. 2nd Int'l Workshop on Computer Aided Verification*.
- Gödel, K.** (1930). *Über die Vollständigkeit des Logikkalküls*. Ph.D. thesis, University of Vienna.
- Gödel, K.** (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38, 173–198.
- Goebel, J., Volk, K., Walker, H. e Gerbault, F.** (1989). Automatic classification of spectra from the infrared astronomical satellite (IRAS). *Astronomy and Astrophysics*, 222, L5–L8.
- Goertzel, B. e Pennachin, C.** (2007). *Artificial General Intelligence*. Springer.
- Gogate, V. e Domingos, P.** (2011). Approximation by quantization. In *UAI-11*.
- Gold, E. M.** (1967). Language identification in the limit. *Information and Control*, 10, 447–474.
- Goldberg, A. V., Kaplan, H. e Werneck, R. F.** (2006). Reach for A*: Efficient point-to-point shortest path algorithms. In *Workshop on algorithm engineering and experiments*.
- Goldberg, Y.** (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10.
- Goldberg, Y., Zhao, K. e Huang, L.** (2013). Efficient implementation of beam-search incremental parsers. In *ACL-13*.
- Goldman, R. e Boddy, M.** (1996). Expressive planning and explicit knowledge. In *AIPS-96*.
- Goldszmidt, M. e Pearl, J.** (1996). Qualitative probabilities for default reasoning, belief revision, and causal modeling. *AIJ*, 84, 57–112.
- Golomb, S. e Baumert, L.** (1965). Backtrack programming. *JACM*, 14, 516–524.
- Golub, G., Heath, M. e Wahba, G.** (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21.
- Gomes, C., Selman, B., Crato, N. e Kautz, H.** (2000). Heavy-tailed phenomena in satisfiability and constraint processing. *JAR*, 24, 67–100.

- Gomes**, C., Kautz, H., Sabharwal, A. e Selman, B. (2008). Satisfiability solvers. In van Harmelen, F., Lifschitz, V. e Porter, B. (a cura di), *Handbook of Knowledge Representation*. Elsevier.
- Gomes**, C. e Selman, B. (2001). Algorithm portfolios. *AIJ*, 126, 43–62.
- Gomes**, C., Selman, B. e Kautz, H. (1998). Boosting combinatorial search through randomization. In *AAAI-98*.
- Gonthier**, G. (2008). Formal proof—The four-color theorem. *Notices of the AMS*, 55, 1382–1393.
- Good**, I. J. (1961). A causal calculus. *British Journal of the Philosophy of Science*, 11, 305–318.
- Good**, I. J. (1965a). The mystery of Go. *New Scientist*, 427, 172–174.
- Good**, I. J. (1965b). Speculations concerning the first ultraintelligent machine. In Alt, F. L. e Rubinoff, M. (a cura di), *Advances in Computers*, Vol. 6. Academic Press.
- Good**, I. J. (1983). *Good Thinking: The Foundations of Probability and Its Applications*. University of Minnesota Press.
- Goodfellow**, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press.
- Goodfellow**, I., Bulatov, Y., Ibarz, J., Arnoud, S. e Shet, V. (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*.
- Goodfellow**, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. e Bengio, Y. (2015a). Generative adversarial nets. In *NeurIPS 27*.
- Goodfellow**, I., Vinyals, O. e Saxe, A. M. (2015b). Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations*.
- Goodman**, J. (2001). A bit of progress in language modeling. Tech. rep., Microsoft Research.
- Goodman**, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K. e Tenenbaum, J. B. (2008). Church: A language for generative models. In *UAI-08*.
- Goodman**, N. (1977). *The Structure of Appearance* (3rd edition). D. Reidel.
- Gopnik**, A. e Glymour, C. (2002). Causal maps and Bayes nets: A cognitive and computational account of theory-formation. In Caruthers, P., Stich, S. e Siegal, M. (a cura di), *The Cognitive Basis of Science*. Cambridge University Press.
- Gordon**, A. D., Graepel, T., Rolland, N., Russo, C., Borgström, J. e Guiver, J. (2014). Tabular: A schema-driven probabilistic programming language. In *POPL-14*.
- Gordon**, A. S. e Hobbs, J. R. (2017). *A Formal Theory of Commonsense Psychology: How People Think People Think*. Cambridge University Press.
- Gordon**, M. J., Milner, A. J. e Wadsworth, C. P. (1979). *Edinburgh LCF*. Springer-Verlag.
- Gordon**, N. (1994). *Bayesian methods for tracking*. Ph.D. thesis, Imperial College.
- Gordon**, N., Salmond, D. J. e Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140, 107–113.
- Gordon**, S. A. (1994). A faster Scrabble move generation algorithm. *Software Practice and Experience*, 24, 219–232.
- Gorry**, G. A. (1968). Strategies for computer-aided diagnosis. *Math. Biosciences*, 2, 293–318.
- Gorry**, G. A., Kassirer, J. P., Essig, A. e Schwartz, W. B. (1973). Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine*, 55, 473–484.
- Gottlob**, G., Leone, N. e Scarcello, F. (1999a). A comparison of structural CSP decomposition methods. In *IJCAI-99*.
- Gottlob**, G., Leone, N. e Scarcello, F. (1999b). Hypertree decompositions and tractable queries. In *PODS-99*.
- Goyal**, Y., Khot, T., Summers-Stay, D., Batra, D. e Parikh, D. (2017). Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *CVPR-17*.
- Grace**, K., Salvatier, J., Dafoe, A., Zhang, B. e Evans, O. (2017). When will AI exceed human performance? Evidence from AI experts. arXiv:1705.08807.
- Graham**, S. L., Harrison, M. A. e Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2, 415–462.
- Grassmann**, H. (1861). *Lehrbuch der Arithmetik*. Th. Chr. Fr. Enslin, Berlin.
- Grayson**, C. J. (1960). Decisions under uncertainty: Drilling decisions by oil and gas operators. Tech. rep., Harvard Business School.
- Green**, B., Wolf, A., Chomsky, C. e Laugherty, K. (1961). BASEBALL: An automatic question answerer. In *Proc. Western Joint Computer Conference*.
- Green**, C. (1969a). Application of theorem proving to problem solving. In *IJCAI-69*.
- Green**, C. (1969b). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D. e Swann, M. (a cura di), *Machine Intelligence 4*. Edinburgh University Press.
- Green**, C. e Raphael, B. (1968). The use of theorem-proving techniques in question-answering systems. In *Proc. 23rd ACM National Conference*.

- Gribkoff**, E., Van den Broeck, G. e Suciu, D. (2014). Understanding the complexity of lifted inference and asymmetric weighted model counting. In *UAI-14*.
- Griffiths**, T. L., Kemp, C. e Tenenbaum, J. B. (2008). Bayesian models of cognition. In Sun, R. (a cura di), *The Cambridge handbook of computational cognitive modeling*. Cambridge University Press.
- Grinstead**, C. e Snell, J. (1997). *Introduction to Probability*. American Mathematical Society.
- Grosz**, B. J. e Stone, P. (2018). A century long commitment to assessing artificial intelligence and its impact on society. *Communications of the ACM*, 61.
- Grove**, W. e Meehl, P. (1996). Comparative efficiency of informal (subjective, impressionistic) and formal (mechanical, algorithmic) prediction procedures: The clinical statistical controversy. *Psychology, Public Policy, and Law*, 2, 293–323.
- Gruber**, T. (2004). Interview of Tom Gruber. *AIS SIGSEMIS Bulletin*, 1.
- Gu**, J. (1989). *Parallel Algorithms and Architectures for Very Fast AI Search*. Ph.D. thesis, Univ. of Utah.
- Guard**, J., Oglesby, F., Bennett, J. e Settle, L. (1969). Semi-automated mathematics. *JACM*, 16, 49–62.
- Guestrin**, C., Koller, D., Gearhart, C. e Kanodia, N. (2003a). Generalizing plans to new environments in relational MDPs. In *IJCAI-03*.
- Guestrin**, C., Koller, D., Parr, R. e Venkataraman, S. (2003b). Efficient solution algorithms for factored MDPs. *JAIR*, 19, 399–468.
- Guestrin**, C., Lagoudakis, M. G. e Parr, R. (2002). Coordinated reinforcement learning. In *ICML-02*.
- Guibas**, L. J., Knuth, D. E. e Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7, 381–413.
- Gulshan**, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316, 2402–2410.
- Gunkel**, D. J. (2018). *Robot Rights*. MIT Press.
- Gunning**, D. (2016). Explainable artificial intelligence (xai). Tech. rep., DARPA.
- Guo**, C., Goldstein, T., Hannun, A. e van der Maaten, L. (2019). Certified data removal from machine learning models. arXiv:1911.03030.
- Gururangan**, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. e Smith, N. A. (2018). Annotation artifacts in natural language inference data. arXiv:1803.02324.
- Guyon**, I., Bennett, K., Cawley, G. C., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A. R. e Viegas, E. (2015). Design of the 2015 ChaLearn AutoML challenge. In *IJCNN-15*.
- Guyon**, I. e Elisseeff, A. (2003). An introduction to variable and feature selection. *JMLR*, 3, 1157–1182.
- Hacking**, I. (1975). *The Emergence of Probability*. Cambridge University Press.
- Hadfield-Menell**, D., Dragan, A. D., Abbeel, P. e Russell, S. J. (2017a). Cooperative inverse reinforcement learning. In *NeurIPS 29*.
- Hadfield-Menell**, D., Dragan, A. D., Abbeel, P. e Russell, S. J. (2017b). The off-switch game. In *IJCAI-17*.
- Hadfield-Menell**, D. e Russell, S. J. (2015). Multitasking: Efficient optimal planning for bandit superprocesses. In *UAI-15*.
- Hailperin**, T. (1984). Probability logic. *Notre Dame J. Formal Logic*, 25, 198–212.
- Hald**, A. (1990). *A History of Probability and Statistics and Their Applications before 1750*. Wiley.
- Hales**, T. (2005). A proof of the Kepler conjecture. *Annals of mathematics*, 162, 1065–1185.
- Hales**, T., Adams, M., Bauer, G., Dang, T. D., Harrison, J., Le Truong, H., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T. T., et al. (2017). A formal proof of the Kepler conjecture. In *Forum of Mathematics, Pi*.
- Halevy**, A. (2007). Dataspaces: A new paradigm for data integration. In *Brazilian Symposium on Databases*.
- Halevy**, A., Norvig, P. e Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, March/April, 8–12.
- Halpern**, J. Y. (1990). An analysis of first-order logics of probability. *AIJ*, 46, 311–350.
- Halpern**, J. Y. (1999). Technical addendum, Cox's theorem revisited. *JAIR*, 11, 429–435.
- Halpern**, J. Y. e Weissman, V. (2008). Using first-order logic to reason about policies. *ACM Transactions on Information and System Security*, 11, 1–41.
- Hammersley**, J. M. e Handscomb, D. C. (1964). *Monte Carlo Methods*. Methuen.
- Han**, J., Pei, J. e Kamber, M. (2011). *Data Mining: Concepts and Techniques*. Elsevier.
- Han**, X. e Boyden, E. (2007). Multiple-color optical activation, silencing, and desynchronization of neural activity, with single-spike temporal resolution. *PLoS One*, e299.

- Handschin, J. E. e Mayne, D. Q.** (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage nonlinear filtering. *Int. J. Control*, 9, 547–559.
- Hans, A., Schneegäss, D., Schäfer, A. M. e Udluft, S.** (2008). Safe exploration for reinforcement learning. In *ESANN*.
- Hansen, E.** (1998). Solving POMDPs by searching in policy space. In *UAI-98*.
- Hansen, E. e Zilberstein, S.** (2001). LAO*: a heuristic search algorithm that finds solutions with loops. *AIJ*, 129, 35–62.
- Hansen, P. e Jaumard, B.** (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44, 279–303.
- Hanski, I. e Cambefort, Y. (a cura di).** (1991). *Dung Beetle Ecology*. Princeton University Press.
- Hansson, O. e Mayer, A.** (1989). Heuristic search as evidential reasoning. In *UAI 5*.
- Haralick, R. M. e Elliott, G. L.** (1980). Increasing tree search efficiency for constraint satisfaction problems. *AIJ*, 14, 263–313.
- Hardin, G.** (1968). The tragedy of the commons. *Science*, 162, 1243–1248.
- Hardt, M., Price, E., Srebro, N., et al.** (2017). Equality of opportunity in supervised learning. In *NeurIPS 29*.
- Harris, T.** (2016). How technology is hijacking your mind—From a magician and Google design ethicist. medium.com/thrive-global/how-technology-hijacks-peoples-minds-from-a-magician-and-google-s-design-ethicist-56d62ef5edf3.
- Harris, Z.** (1954). Distributional structure. *Word*, 10.
- Harrison, J. e March, J. G.** (1984). Decision making and postdecision surprises. *Administrative Science Quarterly*, 29, 26–42.
- Harrow, A. W., Hassidim, A. e Lloyd, S.** (2009). Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103 15, 150502.
- Harsanyi, J.** (1967). Games with incomplete information played by Bayesian players. *Management Science*, 14, 159–182.
- Hart, P. E., Nilsson, N. J. e Raphael, B.** (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2), 100–107.
- Hart, T. P. e Edwards, D. J.** (1961). The tree prune (TP) algorithm. Artificial intelligence project memo, MIT.
- Hartley, H.** (1958). Maximum likelihood estimation from incomplete data. *Biometrics*, 14, 174–194.
- Hartley, R. e Zisserman, A.** (2000). *Multiple view geometry in computer vision*. Cambridge University Press.
- Hashimoto, K., Xiong, C., Tsuruoka, Y. e Socher, R.** (2016). A joint many-task model: Growing a neural network for multiple NLP tasks. arXiv:1611.01587.
- Haslum, P., Botea, A., Helmert, M., Bonet, B. e Koenig, S.** (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI-07*.
- Haslum, P. e Geffner, H.** (2001). Heuristic planning with time and resources. In *Proc. IJCAI-01 Workshop on Planning with Resources*.
- Haslum, P.** (2006). Improving heuristics through relaxed search – An analysis of TP4 and HSP*a in the 2004 planning competition. *JAIR*, 25, 233–267.
- Hastie, T. e Tibshirani, R.** (1996). Discriminant adaptive nearest neighbor classification and regression. In *NeurIPS 8*.
- Hastie, T., Tibshirani, R. e Friedman, J.** (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction* (2nd edition). Springer-Verlag.
- Hastings, W. K.** (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57, 97–109.
- Hatem, M. e Ruml, W.** (2014). Simpler bounded suboptimal search. In *AAAI-14*.
- Haugeland, J.** (1985). *Artificial Intelligence: The Very Idea*. MIT Press.
- Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W. e White, J. L.** (2000). Formal analysis of the remote agent before and after flight. In *Proc. 5th NASA Langley Formal Methods Workshop*.
- Havenstein, H.** (2005). Spring comes to AI winter. *Computer World*, Fe. 14.
- Hawkins, J.** (1961). Self-organizing systems: A review and commentary. *Proc. IRE*, 49, 31–48.
- Hay, N., Russell, S. J., Shimony, S. E. e Tolpin, D.** (2012). Selecting computations: Theory and applications. In *UAI-12*.
- Hayes, P. J.** (1978). The naive physics manifesto. In Michie, D. (a cura di), *Expert Systems in the Microelectronic Age*. Edinburgh University Press.
- Hayes, P. J.** (1979). The logic of frames. In Metzing, D. (a cura di), *Frame Conceptions and Text Understanding*. de Gruyter.
- Hayes, P. J.** (1985a). Naive physics I: Ontology for liquids. In Hobbs, J. R. e Moore, R. C. (a cura di), *Formal Theories of the Commonsense World*, chap. 3. Ablex.

- Hayes, P. J.** (1985b). The second naive physics manifesto. In Hobbs, J. R. e Moore, R. C. (a cura di), *Formal Theories of the Commonsense World*, chap. 1. Ablex.
- Hays, J.** e Efros, A. (2007). Scene completion Using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH)*, 26.
- He, H.**, Bai, Y., Garcia, E. A. e Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*.
- He, K.**, Zhang, X., Ren, S. e Sun, J. (2016). Deep residual learning for image recognition. In *CVPR-16*.
- Heawood, P. J.** (1890). Map colouring theorem. *Quarterly Journal of Mathematics*, 24, 332–338.
- Hebb, D. O.** (1949). *The Organization of Behavior*. Wiley.
- Heckerman, D.** (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. e Lemmer, J. F. (a cura di), *UAI 2*. Elsevier.
- Heckerman, D.** (1991). *Probabilistic Similarity Networks*. MIT Press.
- Heckerman, D.** (1998). A tutorial on learning with Bayesian networks. In Jordan, M. I. (a cura di), *Learning in graphical models*. Kluwer.
- Heckerman, D.**, Geiger, D. e Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical report, Microsoft Research.
- Heess, N.**, Wayne, G., Silver, D., Lillicrap, T., Erez, T. e Tassa, Y. (2016). Learning continuous control policies by stochastic value gradients. In *NeurIPS 28*.
- Heidegger, M.** (1927). *Being and Time*. SCM Press.
- Heinlein, R. A.** (1973). *Time Enough for Love*. Putnam.
- Held, M.** e Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, 18, 1138–1162.
- Helmert, M.** (2001). On the complexity of planning in transportation domains. In *ECP-01*.
- Helmert, M.** (2006). The fast downward planning system. *JAIR*, 26, 191–246.
- Helmert, M.** e Röger, G. (2008). How good is almost perfect? In *AAAI-08*.
- Helmert, M.**, Röger, G. e Karpas, E. (2011). Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS*.
- Hendeby, G.**, Karlsson, R. e Gustafsson, F. (2010). Particle filtering: The need for speed. *EURASIP J. Adv. Sig. Proc., June*.
- Henrion, M.** (1988). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F. e Kanal, L. N. (a cura di), *UAI 2*. Elsevier.
- Henzinger, T. A.** e Sastry, S. (a cura di). (1998). *Hybrid Systems: Computation and Control*. Springer-Verlag.
- Herbrand, J.** (1930). *Recherches sur la Théorie de la Démonstration*. Ph.D. thesis, University of Paris.
- Herbrich, R.**, Minka, T. e Graepel, T. (2007). TrueSkill: A Bayesian skill rating system. In *NeurIPS 19*.
- Hernández-Orallo, J.** (2016). Evaluation in artificial intelligence: From task-oriented to ability-oriented measurement. *Artificial Intelligence Review*, 48, 397–447.
- Hess, C.** e Ostrom, E. (2007). *Understanding Knowledge as a Commons*. MIT Press.
- Hewitt, C.** (1977). Viewing control structures as patterns of passing messages. *AIJ*, 8, 323–364.
- Hewitt, C.** (1969). PLANNER: a language for proving theorems in robots. In *IJCAI-69*.
- Hezaveh, Y. D.**, Levasseur, L. P. e Marshall, P. J. (2017). Fast automated analysis of strong gravitational lenses with convolutional neural networks. *Nature*, 548, 555–557.
- Hierholzer, C.** (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6, 30–32.
- Hilbert, M.** e Lopez, P. (2011). The world's technological capacity to store, communicate, and compute information. *Science*, 332, 60–65.
- Hilgard, E. R.** e Bower, G. H. (1975). *Theories of Learning* (4th edition). Prentice-Hall.
- Hind, M.**, Mehta, S., Mojsilovic, A., Nair, R., Ramamurthy, K. N., Olteanu, A. e Varshney, K. R. (2018). Increasing trust in AI services through supplier's declarations of conformity. arXiv:1808.07261.
- Hintikka, J.** (1962). *Knowledge and Belief*. Cornell University Press.
- Hinton, G. E.** e Anderson, J. A. (1981). *Parallel Models of Associative Memory*. Lawrence Erlbaum.
- Hinton, G. E.** e Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, 1, 495–502.
- Hinton, G. E.** e Sejnowski, T. (1983). Optimal perceptual inference. In *CVPR-83*.
- Hinton, G. E.** e Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. e McClelland, J. L. (a cura di), *Parallel Distributed Processing*. MIT Press.
- Hinton, G. E.** (1987). Learning translation invariant recognition in a massively parallel network. In Goos, G. e Hartmanis, J. (a cura di), *PARLE: Parallel Architectures and Languages Europe*. Springer-Verlag.

- Hinton**, G. E., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. e Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 29, 82–97.
- Hinton**, G. E., Osindero, S. e Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 1527–1554.
- Hirth**, M., Hoßfeld, T. e Tran-Gia, P. (2013). Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms. *Mathematical and Computer Modelling*, 57, 2918–2932.
- Ho**, M. K., Littman, M. L., MacGlashan, J., Cushman, F. e Austerweil, J. L. (2017). Showing versus doing: Teaching by demonstration. In *NeurIPS* 29.
- Ho**, T. K. (1995). Random decision forests. In *Proc. 3rd Int'l Conf. on Document Analysis and Recognition*.
- Hobbs**, J. R. (1990). *Literature and Cognition*. CSLI Press.
- Hobbs**, J. R. e Moore, R. C. (a cura di). (1985). *Formal Theories of the Commonsense World*. Ablex.
- Hochreiter**, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Technische Universität München.
- Hochreiter**, S. e Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9, 1735–1780.
- Hoffmann**, M., Bach, F. R. e Blei, D. M. (2011). Online learning for latent Dirichlet allocation. In *NeurIPS* 23.
- Hoffmann**, J. (2001). FF: The fast-forward planning system. *AIMag*, 22, 57–62.
- Hoffmann**, J. e Brafman, R. I. (2006). Conformant planning via heuristic forward search: A new approach. *AII*, 170, 507–541.
- Hoffmann**, J. e Brafman, R. I. (2005). Contingent planning via heuristic forward search with implicit belief states. In *ICAPS-05*.
- Hoffmann**, J. (2005). Where “ignoring delete lists” works: Local search topology in planning benchmarks. *JAIR*, 24, 685–758.
- Hoffmann**, J. e Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14, 253–302.
- Hoffmann**, J., Sabharwal, A. e Domshlak, C. (2006). Friends or foes? An AI planning perspective on abstraction and search. In *ICAPS-06*.
- Hofleitner**, A., Herring, R., Abbeel, P. e Bayen, A. M. (2012). Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. *IEEE Transactions on Intelligent Transportation Systems*, 13, 1679–1693.
- Hogan**, N. (1985). Impedance control: An approach to manipulation. Parts I, II, and III. *J. Dynamic Systems, Measurement, and Control*, 107, 1–24.
- Hoiem**, D., Efros, A. e Hebert, M. (2007). Recovering surface layout from an image. *IJCV*, 75, 151–172.
- Holland**, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.
- Holland**, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley.
- Holte**, R. C., Fehrer, A., Sharon, G. e Sturtevant, N. R. (2016). Bidirectional search that is guaranteed to meet in the middle. In *AAAI-16*.
- Holzmann**, G. J. (1997). The Spin model checker. *IEEE Transactions on Software Engineering*, 23, 279–295.
- Hood**, A. (1824). Case 4th—28 July 1824 (Mr. Hood’s cases of injuries of the brain). *Phrenological Journal and Miscellany*, 2, 82–94.
- Hooker**, J. (1995). Testing heuristics: We have it all wrong. *J. Heuristics*, 1, 33–42.
- Hoos**, H. H. e Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.
- Hoos**, H. H. e Tsang, E. (2006). Local search methods. In Rossi, F., van Beek, P. e Walsh, T. (a cura di), *Handbook of Constraint Processing*. Elsevier.
- Hopfield**, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *PNAS*, 79, 2554–2558.
- Horn**, A. (1951). On sentences which are true of direct unions of algebras. *JSL*, 16, 14–21.
- Horn**, B. K. P. (1970). Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report, MIT Artificial Intelligence Laboratory.
- Horn**, B. K. P. e Brooks, M. J. (1989). *Shape from Shading*. MIT Press.
- Horn**, K. V. (2003). Constructing a logic of plausible inference: A guide to Cox’s theorem. *IJAR*, 34, 3–24.
- Horning**, J. J. (1969). *A Study of Grammatical Inference*. Ph.D. thesis, Stanford University.
- Horswill**, I. (2000). Functional programming of behavior-based systems. *Autonomous Robots*, 9, 83–93.
- Horvitz**, E. J. (1987). Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proc. Second Annual NASA Research Forum*.
- Horvitz**, E. J. e Barry, M. (1995). Display of information for time-critical decision making. In *UAI-95*.

- Horvitz**, E. J., Breese, J. S., Heckerman, D. e Hovel, D. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *UAI-98*.
- Horvitz**, E. J., Breese, J. S. e Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *IJAR*, 2, 247–302.
- Horvitz**, E. J. e Breese, J. S. (1996). Ideal partition of resources for metareasoning. In *AAAI-96*.
- Hotelling**, H. (1933). Analysis of a complex of statistical variables into principal components. *J. Ed. Psych.*, 24, 417–441.
- Howard**, J. e Gugger, S. (2020). *Deep Learning for Coders with fastai and PyTorch*. O'Reilly.
- Howard**, J. e Ruder, S. (2018). Fine-tuned language models for text classification. arXiv:1801.06146.
- Howard**, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.
- Howard**, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-2*, 22–26.
- Howard**, R. A. (1989). Microrisks for medical decision analysis. *Int. J. Technology Assessment in Health Care*, 5, 357–370.
- Howard**, R. A. e Matheson, J. E. (1984). Influence diagrams. In Howard, R. A. e Matheson, J. E. (a cura di), *Readings on the Principles and Applications of Decision Analysis*. Strategic Decisions Group.
- Howe**, D. (1987). The computational behaviour of Girard's paradox. In *LICS-87*.
- Howson**, C. (2003). Probability and logic. *J. Applied Logic*, 1, 151–165.
- Hsiao**, K., Kaelbling, L. P. e Lozano-Perez, T. (2007). Grasping POMDPs. In *ICRA-07*.
- Hsu**, F.-H. (2004). *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.
- Hsu**, F.-H., Anantharaman, T. S., Campbell, M. S. e Nowatzyk, A. (1990). A grandmaster chess machine. *Scientific American*, 263, 44–50.
- Hu**, J. e Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *ICML-98*.
- Hu**, J. e Wellman, M. P. (2003). Nash Q-learning for general-sum stochastic games. *JMLR*, 4, 1039–1069.
- Huang**, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. J. e Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks. In *AAAI-94*.
- Huang**, T. e Russell, S. J. (1998). Object identification: A Bayesian analysis with application to traffic surveillance. *AIJ*, 103, 1–17.
- Hubel**, D. H. e Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiology*, 160, 106–154.
- Hubel**, D. H. e Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *J. Physiology*, 195, 215–243.
- Hubel**, D. H. (1988). *Eye, Brain, and Vision*. W. H. Freeman.
- Hubel**, D. H. e Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148, 574–591.
- Huddleston**, R. D. e Pullum, G. K. (2002). *The Cambridge Grammar of the English Language*. Cambridge University Press.
- Huffman**, D. A. (1971). Impossible objects as nonsense sentences. In Meltzer, B. e Michie, D. (a cura di), *Machine Intelligence 6*. Edinburgh University Press.
- Hughes**, B. D. (1995). *Random Walks and Random Environments, Vol. 1: Random Walks*. Oxford University Press.
- Hughes**, G. E. e Cresswell, M. J. (1996). *A New Introduction to Modal Logic*. Routledge.
- Huhns**, M. N. e Singh, M. (a cura di). (1998). *Readings in Agents*. Morgan Kaufmann.
- Hume**, D. (1739). *A Treatise of Human Nature* (2nd edition). Republished by Oxford University Press, 1978, Oxford.
- Humphrys**, M. (2008). How my program passed the Turing test. In Epstein, R., Roberts, G. e Beber, G. (a cura di), *Parsing the Turing Test*. Springer.
- Hunsberger**, L. e Grosz, B. J. (2000). A combinatorial auction for collaborative planning. In *Int. Conference on Multi-Agent Systems*.
- Hunt**, W. e Brock, B. (1992). A formal HDL and its use in the FM9001 verification. *Phil. Trans. Roy. Soc.*, 339.
- Hunter**, L. e States, D. J. (1992). Bayesian classification of protein structure. *IEEE Expert*, 7, 67–75.
- Hur**, C.-K., Nori, A. V., Rajamani, S. K. e Samuel, S. (2014). Slicing probabilistic programs. In *PLDI-14*.
- Hurst**, M. (2000). *The Interpretation of Text in Tables*. Ph.D. thesis, Edinburgh.
- Hurwicz**, L. (1973). The design of mechanisms for resource allocation. *American Economic Review Papers and Proceedings*, 63, 1–30.

- Huth**, M. e Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning About Systems* (2nd edition). Cambridge University Press.
- Huttenlocher**, D. e Ullman, S. (1990). Recognizing solid objects by alignment with an image. *IJCV*, 5, 195–212.
- Hutter**, F., Kotthoff, L. e Vanschoren, J. (2019). *Automated Machine Learning*. Springer.
- Huygens**, C. (1657). De ratiociniis in ludo aleae. In van Schooten, F. (a cura di), *Exercitionum Mathematicorum*. Elsevirii, Amsterdam. Translated into English by John Arbuthnot (1692).
- Huyn**, N., Dechter, R. e Pearl, J. (1980). Probabilistic analysis of the complexity of A*. *AIJ*, 15, 241–254.
- Huynh**, V. A. e Roy, N. (2009). icLQG: Combining local and global optimization for control in information space. In *ICRA-09*.
- Hwa**, R. (1998). An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *ACL-98*.
- Hwang**, C. H. e Schubert, L. K. (1993). EL: A formal, yet natural, comprehensive knowledge representation. In *AAAI-93*.
- Hyafil**, L. e Rivest, R. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5, 15–17.
- Jeong**, S. e Shoham, Y. (2005). Marginal contribution nets: A compact representation scheme for coalitional games. In *Proc. Sixth ACM Conference on Electronic Commerce (EC'05)*.
- Ingerman**, P. Z. (1967). Panini–Backus form suggested. *CACM*, 10, 137.
- Intille**, S. e Bobick, A. (1999). A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*.
- Ioffe**, S. e Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.
- Irpan**, A. (2018). Deep reinforcement learning doesn't work yet. www.alexirpan.com/2018/02/14/rl-hard.html.
- Isard**, M. e Blake, A. (1996). Contour tracking by stochastic propagation of conditional density. In *ECCV-96*.
- Isola**, P., Zhu, J.-Y., Zhou, T. e Efros, A. (2017). Image-to-image translation with conditional adversarial networks. In *CVPR-17*.
- Jaakkola**, T. e Jordan, M. I. (1996). Computing upper and lower bounds on likelihoods in intractable networks. In *UAI-96*.
- Jacobson**, D. H. e Mayne, D. Q. (1970). *Differential Dynamic Programming*. North-Holland.
- Jaderberg**, M., Czarnecki, W. M., Dunning, I., Marrs, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364, 859–865.
- Jaderberg**, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C. e Kavukcuoglu, K. (2017). Population based training of neural networks. arXiv:1711.09846.
- Jaffar**, J. e Lassez, J.-L. (1987). Constraint logic programming. In *Proc. Fourteenth ACM POPL Conference*. Association for Computing Machinery.
- Jaffar**, J., Michaylov, S., Stuckey, P. J. e Yap, R. H. C. (1992). The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14, 339–395.
- Jain**, D., Barthels, A. e Beetz, M. (2010). Adaptive Markov logic networks: Learning statistical relational models with dynamic parameters. In *ECAI-10*.
- Jain**, D., Kirchlechner, B. e Beetz, M. (2007). Extending Markov logic to model probability distributions in relational domains. In *30th Annual German Conference on AI (KI)*.
- James**, G., Witten, D., Hastie, T. e Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer-Verlag.
- Jarrett**, K., Kavukcuoglu, K., Ranzato, M. e LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *ICCV-09*.
- Jaynes**, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge Univ. Press.
- Jeffrey**, R. C. (1983). *The Logic of Decision* (2nd edition). University of Chicago Press.
- Jeffreys**, H. (1948). *Theory of Probability*. Oxford.
- Jelinek**, F. (1976). Continuous speech recognition by statistical methods. *Proc. IEEE*, 64, 532–556.
- Jelinek**, F. e Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice*.
- Jennings**, H. S. (1906). *Behavior of the Lower Organisms*. Columbia University Press.
- Jenniskens**, P., Betlem, H., Betlem, J. e Barifaijo, E. (1994). The Mbale meteorite shower. *Meteoritics*, 29, 246–254.
- Jensen**, F. V. (2007). *Bayesian Networks and Decision Graphs*. Springer-Verlag.

- Ji**, Z., Lipton, Z. C. e Elkan, C. (2014). Differential privacy and machine learning: A survey and review. arXiv:1412.7584.
- Jiang**, H. e Nachum, O. (2019). Identifying and correcting label bias in machine learning. arXiv:1901.04966.
- Jimenez**, P. e Torras, C. (2000). An efficient algorithm for searching implicit AND/OR graphs with cycles. *AIJ*, 124, 1–30.
- Joachims**, T. (2001). A statistical learning model of text classification with support vector machines. In *SIGIR-01*.
- Johnson**, M. (1998). PCFG models of linguistic tree representations. *Comput. Linguist.*, 24, 613–632.
- Johnson**, W. W. e Story, W. E. (1879). Notes on the “15” puzzle. *American Journal of Mathematics*, 2, 397–404.
- Johnston**, M. D. e Adorf, H.-M. (1992). Scheduling with neural networks: The case of the Hubble space telescope. *Computers and Operations Research*, 19, 209–240.
- Jonathan**, P. J. Y., Fung, C. C. e Wong, K. W. (2009). Devious chatbots-interactive malware with a plot. In *FIRA RoboWorld Congress*.
- Jones**, M. e Love, B. C. (2011). Bayesian fundamentalism or enlightenment? On the explanatory status and theoretical contributions of Bayesian models of cognition. *BBS*, 34, 169–231.
- Jones**, R. M., Laird, J. e Nielsen, P. E. (1998). Automated intelligent pilots for combat flight simulation. In *AAAI-98*.
- Jones**, R., McCallum, A., Nigam, K. e Riloff, E. (1999). Bootstrapping for text learning tasks. In *Proc. IJCAI-99 Workshop on Text Mining: Foundations, Techniques, and Applications*.
- Jones**, T. (2007). *Artificial Intelligence: A Systems Approach*. Infinity Science Press.
- Jonsson**, A., Morris, P., Muscettola, N., Rajan, K. e Smith, B. (2000). Planning in interplanetary space: Theory and practice. In *AIPS-00*.
- Jordan**, M. I. (2005). Dirichlet processes, Chinese restaurant processes and all that. Tutorial presentation at the NeurIPS Conference.
- Jordan**, M. I. (1986). Serial order: A parallel distributed processing approach. Tech. rep., UCSD Institute for Cognitive Science.
- Jordan**, M. I., Ghahramani, Z., Jaakkola, T. e Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37, 183–233.
- Jouannaud**, J.-P. e Kirchner, C. (1991). Solving equations in abstract algebras: A rule-based survey of unification. In Lassez, J.-L. e Plotkin, G. (a cura di), *Computational Logic*. MIT Press.
- Joulin**, A., Grave, E., Bojanowski, P. e Mikolov, T. (2016). Bag of tricks for efficient text classification. arXiv:1607.01759.
- Jouppi**, N. P., Young, C., Patil, N., Patterson, D. A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *ACM/IEEE 44th International Symposium on Computer Architecture*.
- Joy**, B. (2000). Why the future doesn’t need us. *Wired*, 8.
- Jozefowicz**, R., Vinyals, O., Schuster, M., Shazeer, N. e Wu, Y. (2016). Exploring the limits of language modeling. arXiv:1602.02410.
- Jozefowicz**, R., Zaremba, W. e Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *ICML-15*.
- Juels**, A. e Wattenberg, M. (1996). Stochastic hill-climbing as a baseline method for evaluating genetic algorithms. In *NeurIPS* 8.
- Julesz**, B. (1971). *Foundations of Cyclopean Perception*. University of Chicago Press.
- Julian**, K. D., Kochenderfer, M. J. e Owen, M. P. (2018). Deep neural network compression for aircraft collision avoidance systems. arXiv:1810.04240.
- Juliani**, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M. e Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv:1809.02627.
- Junker**, U. (2003). The logic of ilog (j)configurator: Combining constraint programming with a description logic. In *Proc. IJCAI-03 Configuration Workshop*.
- Jurafsky**, D. e Martin, J. H. (2020). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (3rd edition). Prentice-Hall.
- Kadane**, J. B. e Simon, H. A. (1977). Optimal strategies for a class of constrained sequential problems. *Annals of Statistics*, 5, 237–255.
- Kadane**, J. B. e Larkey, P. D. (1982). Subjective probability and the theory of games. *Management Science*, 28, 113–120.
- Kaelbling**, L. P., Littman, M. L. e Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *AIJ*, 101, 99–134.

- Kaelbling**, L. P. e Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6, 35–48.
- Kager**, R. (1999). *Optimality Theory*. Cambridge University Press.
- Kahn**, H. e Marshall, A. W. (1953). Methods of reducing sample size in Monte Carlo computations. *Operations Research*, 1, 263–278.
- Kahn**, H. (1950a). Random sampling (Monte Carlo) techniques in neutron attenuation problems—I. *Nucleonics*, 6, 27–passim.
- Kahn**, H. (1950b). Random sampling (Monte Carlo) techniques in neutron attenuation problems—II. *Nucleonics*, 6, 60–65.
- Kahneman**, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Kahneman**, D., Slovic, P. e Tversky, A. (a cura di). (1982). *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press.
- Kahneman**, D. e Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica*, 47, 263–291.
- Kaindl**, H. e Khorsand, A. (1994). Memory-bounded bidirectional search. In *AAAI-94*.
- Kalman**, R. (1960). A new approach to linear filtering and prediction problems. *J. Basic Engineering*, 82, 35–46.
- Kambhampati**, S. (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10, 213–244.
- Kanade**, T., Thorpe, C. e Whittaker, W. (1986). Autonomous land vehicle project at CMU. In *ACM Fourteenth Annual Conference on Computer Science*.
- Kanal**, E. (2017). Machine learning in cybersecurity. CMU SEI Blog, insights.sei.cmu.edu/sei.blog/2017/06/machine-learning-in-cybersecurity.html.
- Kanazawa**, A., Black, M., Jacobs, D. e Malik, J. (2018a). End-to-end recovery of human shape and pose. In *CVPR-18*.
- Kanazawa**, A., Tulsiani, M., Efros, A. e Malik, J. (2018b). Learning category-specific mesh reconstruction from image collections. In *ECCV-18*.
- Kanazawa**, K., Koller, D. e Russell, S. J. (1995). Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI-95*.
- Kang**, S. M. e Wildes, R. P. (2016). Review of action recognition and detection methods. arXiv:1610.06906.
- Kanter**, J. M. e Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. In *Proc. IEEE Int'l Conf. on Data Science and Advanced Analytics*.
- Kantorovich**, L. V. (1939). Mathematical methods of organizing and planning production. Published in translation in *Management Science*, 6(4), 366–422, 1960.
- Kaplan**, D. e Montague, R. (1960). A paradox regained. *Notre Dame Formal Logic*, 1, 79–90.
- Karaboga**, D. e Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of global optimization*, 39, 459–471.
- Karamchandani**, A., Bjerager, P. e Cornell, C. A. (1989). Adaptive importance sampling. In *Proc. Fifth International Conference on Structural Safety and Reliability*.
- Karmarkar**, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4, 373–395.
- Karp**, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. e Thatcher, J. W. (a cura di), *Complexity of Computer Computations*. Plenum.
- Karpathy**, A. (2015). The unreasonable effectiveness of recurrent neural networks. Andrej Karpathy Blog karpathy.github.io/2015/05/21/rnn-effectiveness/.
- Karpathy**, A. e Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *CVPR-15*.
- Karras**, T., Aila, T., Laine, S. e Lehtinen, J. (2017). Progressive growing of GANs for improved quality, stability, and variation. arXiv:1710.10196.
- Karsch**, K., Hedau, V., Forsyth, D. e Hoiem, D. (2011). Rendering synthetic objects into legacy photographs. In *SIGGRAPH Asia*.
- Kartam**, N. A. e Levitt, R. E. (1990). A constraint-based approach to construction planning of multi-story buildings. In *Expert Planning Systems*. Institute of Electrical Engineers.
- Kasami**, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep., Air Force Cambridge Research Laboratory.
- Katehakis**, M. N. e Veinott, A. F. (1987). The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12, 185–376.
- Katz**, B. (1997). Annotating the world wide web using natural language. In *RIA'97*.
- Kaufmann**, M., Manolios, P. e Moore, J. S. (2000). *Computer-Aided Reasoning: An Approach*. Kluwer.
- Kautz**, H. (2006). Deconstructing planning as satisfiability. In *AAAI-06*.

- Kautz**, H., McAllester, D. A. e Selman, B. (1996). Encoding plans in propositional logic. In *KR-96*.
- Kautz**, H. e Selman, B. (1992). Planning as satisfiability. In *ECAI-92*.
- Kautz**, H. e Selman, B. (1998). BLACKBOX: A new approach to the application of theorem proving to problem solving. Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search.
- Kavraki**, L., Svestka, P., Latombe, J.-C. e Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12, 566–580.
- Kazemi**, S. M., Kimmig, A., Van den Broeck, G. e Poole, D. (2017). New liftable classes for first-order probabilistic inference. In *NeurIPS 29*.
- Kearns**, M. (1990). *The Computational Complexity of Machine Learning*. MIT Press.
- Kearns**, M., Mansour, Y. e Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. In *NeurIPS 12*.
- Kearns**, M. e Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *ICML-98*.
- Kearns**, M. e Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press.
- Kearns**, M. (1988). Thoughts on hypothesis boosting.
- Kearns**, M., Mansour, Y. e Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49, 193–208.
- Kebeasy**, R. M., Hussein, A. I. e Dahy, S. A. (1998). Discrimination between natural earthquakes and nuclear explosions using the Aswan Seismic Network. *Annali di Geofisica*, 41, 127–140.
- Keeney**, R. L. (1974). Multiplicative utility functions. *Operations Research*, 22, 22–34.
- Keeney**, R. L. e Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley.
- Kelley**, H. J. (1960). Gradient theory of optimal flight paths. *ARS Journal*, 30, 947–954.
- Kemp**, M. (a cura di). (1989). *Leonardo on Painting: An Anthology of Writings*. Yale University Press.
- Kempe**, A. B. (1879). On the geographical problem of the four-colors. *American Journal of Mathematics*, 2, 193–200.
- Kephart**, J. O. e Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, 36, 41–50.
- Kersting**, K., Raedt, L. D. e Kramer, S. (2000). Interpreting Bayesian logic programs. In *Proc. AAAI-00 Workshop on Learning Statistical Models from Relational Data*.
- Keskar**, N. S., McCann, B., Varshney, L., Xiong, C. e Socher, R. (2019). CTRL: A conditional transformer language model for controllable generation. arXiv:1909.
- Keynes**, J. M. (1921). *A Treatise on Probability*. Macmillan.
- Khare**, R. (2006). Microformats: The next (small) thing on the semantic web. *IEEE Internet Computing*, 10, 68–75.
- Khatib**, O. (1986). Real-time obstacle avoidance for robot manipulator and mobile robots. *Int. J. Robotics Research*, 5, 90–98.
- Kim**, B., Khanna, R. e Koyejo, O. O. (2017). Examples are not enough, learn to criticize! Criticism for interpretability. In *NeurIPS 29*.
- Kim**, J. H. (1983). *CONVINCE: A Conversational Inference Consolidation Engine*. Ph.D. thesis, Department of Computer Science, UCLA.
- Kim**, J. H. e Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *IJCAI-83*.
- Kim**, J.-H., Lee, C.-H., Lee, K.-H. e Kuppuswamy, N. (2007). Evolving personality of a genetic robot in ubiquitous environment. In *Proc. 16th IEEE International Symposium on Robot and Human Interactive Communication*.
- Kim**, T. W. (2018). Explainable artificial intelligence (XAI), the goodness criteria and the grasp-ability test. arXiv:1810.09598.
- Kingma**, D. P. e Welling, M. (2013). Auto-encoding variational Bayes. arXiv:1312.6114.
- Kirk**, D. E. (2004). *Optimal Control Theory: An Introduction*. Dover.
- Kirkpatrick**, S., Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.
- Kisynski**, J. e Poole, D. (2009). Lifted aggregation in directed first-order probabilistic models. In *IJCAI-09*.
- Kitaev**, N., Kaiser, L. e Levskaya, A. (2020). Reformer: The efficient transformer. arXiv:2001.04451.
- Kitaev**, N. e Klein, D. (2018). Constituency parsing with a self-attentive encoder. arXiv:1805.01052.
- Kitani**, K. M., abd James Andrew Bagnell, B. D. Z. e Hebert, M. (2012). Activity forecasting. In *ECCV-12*.
- Kitano**, H., Asada, M., Kuniyoshi, Y., Noda, I. e Osawa, E. (1997). RoboCup: The robot world cup initiative. In *Proc. First International Conference on Autonomous Agents*.
- Kjaerulff**, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *UAI-92*.
- Klarman**, H. E., Francis, J. e Rosenthal, G. D. (1968). Cost effectiveness analysis applied to the treatment of chronic renal disease. *Medical Care*, 6, 48–54.
- Klein**, D. e Manning, C. (2001). Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *ACL-01*.

- Klein**, D. e Manning, C. (2003). A* parsing: Fast exact Viterbi parse selection. In *HLT-NAACL-03*.
- Kleinberg**, J. M., Mullainathan, S. e Raghavan, M. (2016). Inherent trade-offs in the fair determination of risk scores. arXiv:1609.05807.
- Klemperer**, P. (2002). What really matters in auction design. *J. Economic Perspectives*, 16.
- Kneser**, R. e Ney, H. (1995). Improved backing-off for M-gram language modeling. In *ICASSP-95*.
- Knoblock**, C. A. (1991). Search reduction in hierarchical problem solving. In *AAAI-91*.
- Knuth**, D. E. (1964). Representing numbers using only one 4. *Mathematics Magazine*, 37, 308–310.
- Knuth**, D. E. (1975). An analysis of alpha–beta pruning. *AIJ*, 6, 293–326.
- Knuth**, D. E. (2015). *The Art of Computer Programming*, Vol. 4, Fascicle 6: Satisfiability. Addison-Wesley.
- Knuth**, D. E. e Bendix, P. B. (1970). Simple word problems in universal algebras. In Leech, J. (a cura di), *Computational Problems in Abstract Algebra*. Pergamon.
- Kober**, J., Bagnell, J. A. e Peters, J. (2013). Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32, 1238–1274.
- Koch**, C. (2019). *The Feeling of Life Itself*. MIT Press.
- Kochenderfer**, M. J. (2015). *Decision Making Under Uncertainty: Theory and Application*. MIT Press.
- Kocsis**, L. e Szepesvari, C. (2006). Bandit-based Monte-Carlo planning. In *ECML-06*.
- Koditschek**, D. (1987). Exact robot navigation by means of potential functions: Some topological considerations. In *ICRA-87*.
- Koehn**, P. (2009). *Statistical Machine Translation*. Cambridge University Press.
- Koelsch**, S. e Siebel, W. A. (2005). Towards a neural basis of music perception. *Trends in Cognitive Sciences*, 9, 578–584.
- Koenderink**, J. J. (1990). *Solid Shape*. MIT Press.
- Koenderink**, J. J. e van Doorn, A. J. (1991). Affine structure from motion. *J. Optical Society of America A*, 8, 377–385.
- Koenig**, S. (1991). Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's report, Computer Science Division, University of California, Berkeley.
- Koenig**, S. (2000). Exploring unknown environments with real-time search or reinforcement learning. In *NeurIPS 12*.
- Koenig**, S. (2001). Agent-centered search. *AIMag*, 22, 109–131.
- Koenig**, S. e Likhachev, M. (2002). D* Lite. *AAAI-15*, 15.
- Koenig**, S., Likhachev, M. e Furcy, D. (2004). Lifelong planning A*. *AIJ*, 155, 93–146.
- Kolesky**, D. B., Truby, R. L., Gladman, A. S., Busbee, T. A., Homan, K. A. e Lewis, J. A. (2014). 3D bioprinting of vascularized, heterogeneous cell-laden tissue constructs. *Advanced Materials*, 26, 3124–3130.
- Koller**, D., Meggido, N. e von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour*, 14, 247–259.
- Koller**, D. e Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *AIJ*, 94, 167–215.
- Koller**, D. e Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI-98*.
- Koller**, D. e Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
- Koller**, D., McAllester, D. A. e Pfeffer, A. (1997). Effective Bayesian inference for stochastic programs. In *AAAI-97*.
- Koller**, D. e Parr, R. (2000). Policy iteration for factored MDPs. In *UAI-00*.
- Koller**, D. e Sahami, M. (1997). Hierarchically classifying documents using very few words. In *ICML-97*.
- Kolmogorov**, A. N. (1941). Interpolation und extrapolation von stationären zufälligen folgen. *Bulletin of the Academy of Sciences of the USSR, Ser. Math.* 5, 3–14.
- Kolmogorov**, A. N. (1950). *Foundations of the Theory of Probability*. Chelsea.
- Kolmogorov**, A. N. (1963). On tables of random numbers. *Sankhya, the Indian Journal of Statistics: Series A*, 25(4), 369–376.
- Kolmogorov**, A. N. (1965). Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1, 1–7.
- Kolter**, J. Z., Abbeel, P. e Ng, A. Y. (2008). Hierarchical apprenticeship learning, with application to quadruped locomotion. In *NeurIPS 20*.
- Kondrak**, G. e van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *AIJ*, 89, 365–387.
- Konečný**, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T. e Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492.
- Konolige**, K. (1997). COLBERT: A language for reactive control in Saphira. In *Künstliche Intelligenz: Advances in Artificial Intelligence*, LNAI.
- Konolige**, K. (2004). Large-scale map-making. In *AAAI-04*.

- Konolige**, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D. e Pao, Y.-H. (a cura di), *Machine Intelligence 10*. Ellis Horwood.
- Konolige**, K. (1994). Easy to be hard: Difficult problems for greedy algorithms. In *KR-94*.
- Koopmans**, T. C. (1972). Representation of preference orderings over time. In McGuire, C. B. e Radner, R. (a cura di), *Decision and Organization*. Elsevier.
- Korb**, K. B. e Nicholson, A. (2010). *Bayesian Artificial Intelligence*. CRC Press.
- Korf**, R. E. (1985a). Depth-first iterative-deepening: an optimal admissible tree search. *AIJ*, 27, 97–109.
- Korf**, R. E. (1985b). Iterative-deepening A*: An optimal admissible tree search. In *IJCAI-85*.
- Korf**, R. E. (1987). Planning as search: A quantitative approach. *AIJ*, 33, 65–88.
- Korf**, R. E. (1990). Real-time heuristic search. *AIJ*, 42, 189–212.
- Korf**, R. E. (1993). Linear-space best-first search. *AIJ*, 62, 41–78.
- Korf**, R. E. e Chickering, D. M. (1996). Best-first minimax search. *AIJ*, 84, 299–337.
- Korf**, R. E. e Felner, A. (2002). Disjoint pattern database heuristics. *AIJ*, 134, 9–22.
- Korf**, R. E. e Zhang, W. (2000). Divide-and-conquer frontier search applied to optimal sequence alignment. In *AAAI-00*.
- Korf**, R. E. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI-97*.
- Korf**, R. E. e Reid, M. (1998). Complexity analysis of admissible heuristic search. In *AAAI-98*.
- Koutsoupias**, E. e Papadimitriou, C. H. (1992). On the greedy algorithm for satisfiability. *Information Processing Letters*, 43, 53–55.
- Kovacs**, D. L. (2011). BNF definition of PDDL3.1. Unpublished manuscript from the IPC-2011 website.
- Kowalski**, R. (1974). Predicate logic as a programming language. In *Proc. IFIP Congress*.
- Kowalski**, R. (1979). *Logic for Problem Solving*. Elsevier.
- Kowalski**, R. (1988). The early years of logic programming. *CACM*, 31, 38–43.
- Kowalski**, R. e Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4, 67–95.
- Koza**, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.
- Koza**, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.
- Koza**, J. R., Bennett, F. H., Andre, D. e Keane, M. A. (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann.
- Krakovna**, V. (2018). Specification gaming examples in AI.
- Kraska**, T., Beutel, A., Chi, E. H., Dean, J. e Polyzotis, N. (2017). The case for learned index structures. arXiv:1712.01208.
- Kraus**, S. (2001). *Strategic Negotiation in Multiagent Environments*. MIT Press.
- Kraus**, S., Ephrati, E. e Lehmann, D. (1991). Negotiation in a non-cooperative environment. *AIJ*, 3, 255–281.
- Krause**, A. e Guestrin, C. (2005). Optimal nonmyopic value of information in graphical models: Efficient algorithms and theoretical limits. In *IJCAI-05*.
- Krause**, A. e Guestrin, C. (2009). Optimal value of information in graphical models. *JAIR*, 35, 557–591.
- Krause**, A., McMahan, B., Guestrin, C. e Gupta, A. (2008). Robust submodular observation selection. *JMLR*, 9, 2761–2801.
- Kripke**, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16, 83–94.
- Krishna**, V. (2002). *Auction Theory*. Academic Press.
- Krishnamurthy**, V. (2016). *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press.
- Krishnanand**, K. e Ghose, D. (2009). Glowworm swarm optimisation: A new method for optimising multi-modal functions. *International Journal of Computational Intelligence Studies*, 1, 93–119.
- Krizhevsky**, A., Sutskever, I. e Hinton, G. E. (2013). ImageNet classification with deep convolutional neural networks. In *NeurIPS 25*.
- Krogh**, A., Brown, M., Mian, I. S., Sjolander, K. e Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *J. Molecular Biology*, 235, 1501–1531.
- Krogh**, A. e Hertz, J. A. (1992). A simple weight decay can improve generalization. In *NeurIPS 4*.
- Kruppa**, E. (1913). Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung. *Sitz.Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. IIa*, 122, 1939–1948.
- Kübler**, S., McDonald, R. e Nivre, J. (2009). *Dependency Parsing*. Morgan & Claypool.
- Kuffner**, J. J. e LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *ICRA-00*.
- Kuhn**, H. W. (1953). Extensive games and the problem of information. In Kuhn, H. W. e Tucker, A. W. (a cura di), *Contributions to the Theory of Games II*. Princeton University Press.

- Kuhn, H. W.** (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2, 83–97.
- Kuipers, B. J.** (1985). Qualitative simulation. In Bobrow, D. (a cura di), *Qualitative Reasoning About Physical Systems*. MIT Press.
- Kuipers, B. J.** e Levitt, T. S. (1988). Navigation and mapping in large-scale space. *AIMag*, 9, 25–43.
- Kuipers, B. J.** (2001). Qualitative simulation. In Meyers, R. A. (a cura di), *Encyclopedia of Physical Science and Technology*. Academic Press.
- Kulkarni, T.**, Kohli, P., Tenenbaum, J. B. e Mansinghka, V. K. (2015). Picture: A probabilistic programming language for scene perception. In *CVPR-15*.
- Kumar, P. R.** e Varaiya, P. (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall.
- Kumar, S.** (2017). A survey of deep learning methods for relation extraction. arXiv:1705.03645.
- Kumar, V.** e Kanal, L. N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal, L. N. e Kumar, V. (a cura di), *Search in Artificial Intelligence*. Springer-Verlag.
- Kurien, J.**, Nayak, P. e Smith, D. E. (2002). Fragment-based conformant planning. In *AIPS-02*.
- Kurth, T.**, Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E. H., Mahesh, A., Matheson, M., Deslippe, J., Fatica, M., Prabhat e Houston, M. (2018). Exascale deep learning for climate analytics. arXiv:1810.01993.
- Kurzweil, R.** (2005). *The Singularity is Near*. Viking.
- Kwok, C.**, Etzioni, O. e Weld, D. S. (2001). Scaling question answering to the web. In *Proc. 10th International Conference on the World Wide Web*.
- La Mettrie, J. O.** (1748). *L'homme machine*. E. Luzac, Leyde, France.
- La Mura, P.** e Shoham, Y. (1999). Expected utility networks. In *UAI-99*.
- Laborie, P.** (2003). Algorithms for propagating resource constraints in AI planning and scheduling. *AIJ*, 143, 151–188.
- Ladkin, P.** (1986a). Primitives and units for time specification. In *AAAI-86*.
- Ladkin, P.** (1986b). Time representation: a taxonomy of interval relations. In *AAAI-86*.
- Lafferty, J.**, McCallum, A. e Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML-01*.
- Lai, T. L.** e Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6, 4–22.
- Laird, J.**, Newell, A. e Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *AIJ*, 33, 1–64.
- Laird, J.**, Rosenbloom, P. S. e Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1, 11–46.
- Laird, J.** (2008). Extending the Soar cognitive architecture. In *Artificial General Intelligence Conference*.
- Lake, B.**, Salakhutdinov, R. e Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350, 1332–1338.
- Lakoff, G.** (1987). *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. University of Chicago Press.
- Lakoff, G.** e Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.
- Lakoff, G.** e Johnson, M. (1999). *Philosophy in the Flesh : The Embodied Mind and Its Challenge to Western Thought*. Basic Books.
- Lam, J.** e Greenspan, M. (2008). Eye-in-hand visual servoing for accurate shooting in pool robotics. In *5th Canadian Conference on Computer and Robot Vision*.
- Lamarck, J. B.** (1809). *Philosophie zoologique*. Chez Dentu et L'Auteur, Paris.
- Lample, G.** e Conneau, A. (2019). Cross-lingual language model pretraining. arXiv:1901.07291.
- Landhuis, E.** (2004). Lifelong debunker takes on arbiter of neutral choices: Magician-turned-mathematician uncovers bias in a flip of a coin. *Stanford Report*, June 7.
- Langdon, W.** e Poli, R. (2002). *Foundations of Genetic Programming*. Springer.
- Langton, C.** (a cura di). (1995). *Artificial Life*. MIT Press.
- LaPaugh, A. S.** (2010). Algorithms and theory of computation handbook. In Atallah, M. J. e Blanton, M. (a cura di), *VLSI Layout Algorithms*. Chapman & Hall/CRC.
- Laplace, P.** (1816). *Essai philosophique sur les probabilités* (3rd edition). Courcier Imprimeur, Paris.
- Larochelle, H.** e Murray, I. (2011). The neural autoregressive distribution estimator. In *AISTATS-11*.
- Larson, S. C.** (1931). The shrinkage of the coefficient of multiple correlation. *J. Educational Psychology*, 22, 45–55.

- Laskey**, K. B. (1995). Sensitivity analysis for probability assessments in Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics*, 25, 901–909.
- Laskey**, K. B. (2008). MEBN: A language for first-order Bayesian knowledge bases. *AIJ*, 172, 140–178.
- Latombe**, J.-C. (1991). *Robot Motion Planning*. Kluwer.
- Lauritzen**, S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19, 191–201.
- Lauritzen**, S., Dawid, A. P., Larsen, B. e Leimer, H. (1990). Independence properties of directed Markov fields. *Networks*, 20, 491–505.
- Lauritzen**, S. e Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society, B* 50, 157–224.
- Lauritzen**, S. e Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17, 31–57.
- LaValle**, S. (2006). *Planning Algorithms*. Cambridge University Press.
- Lawler**, E. L., Lenstra, J. K., Kan, A. e Shmoys, D. B. (1992). *The Travelling Salesman Problem*. Wiley Interscience.
- Lawler**, E. L., Lenstra, J. K., Kan, A. e Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In Graves, S. C., Zipkin, P. H. e Kan, A. H. G. R. (a cura di), *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science, Volume 4*. North-Holland.
- Lawler**, E. L. e Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14, 699–719.
- Lazanas**, A. e Latombe, J.-C. (1992). Landmark-based robot navigation. In *AAAI-92*.
- Le**, T. A., Baydin, A. G. e Wood, F. (2017). Inference compilation and universal probabilistic programming. In *AISTATS-17*.
- Lebedev**, M. A. e Nicolelis, M. A. (2006). Brain-machine interfaces: Past, present and future. *Trends in Neurosciences*, 29, 536–546.
- Lecoutre**, C. (2009). *Constraint Networks: Techniques and Algorithms*. Wiley-IEEE Press.
- LeCun**, Y., Denker, J. e Solla, S. (1990). Optimal brain damage. In *NeurIPS 2*.
- LeCun**, Y., Jackel, L., Boser, B. e Denker, J. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27, 41–46.
- LeCun**, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard, P. e Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Int. Conference on Artificial Neural Networks*.
- LeCun**, Y., Bengio, Y. e Hinton, G. E. (2015). Deep learning. *Nature*, 521, 436–444.
- Lee**, D., Seo, H. e Jung, M. W. (2012). Neural basis of reinforcement learning and decision making. *Annual Review of Neuroscience*, 35, 287–308.
- Lee**, K.-F. (2018). *AI Superpowers: China, Silicon Valley, and the New World Order*. Houghton Mifflin.
- Leech**, G., Rayson, P. e Wilson, A. (2001). *Word Frequencies in Written and Spoken English: Based on the British National Corpus*. Longman.
- Legendre**, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. Chez Firmin Didot, Paris.
- Lehmann**, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S. e Bizer, C. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6, 167–195.
- Lehrer**, J. (2009). *How We Decide*. Houghton Mifflin.
- Leike**, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L. e Legg, S. (2017). AI safety gridworlds. arXiv:1711.09883.
- Lelis**, L., Arfaee, S. J., Zilles, S. e Holte, R. C. (2012). Learning heuristic functions faster by using predicted solution costs. In *Proc. Fifth Annual Symposium on Combinatorial Search*.
- Lenat**, D. B. (1975). BEINGS: Knowledge as interacting experts. In *IJCAI-75*.
- Lenat**, D. B. e Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.
- Leonard**, H. S. e Goodman, N. (1940). The calculus of individuals and its uses. *JSL*, 5, 45–55.
- Leonard**, J. e Durrant-Whyte, H. (1992). *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer.
- Lepage**, G. P. (1978). A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27, 192–203.
- Lerner**, U. (2002). *Hybrid Bayesian Networks for Reasoning About Complex Systems*. Ph.D. thesis, Stanford University.
- Leśniewski**, S. (1916). Podstawy ogólnej teorii mnogości. Popławski.

- Lesser**, V. R. e Corkill, D. D. (1988). The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. In Engelmore, R. e Morgan, T. (a cura di), *Blackboard Systems*. Addison-Wesley.
- Letz**, R., Schumann, J., Bayerl, S. e Bibel, W. (1992). SETHEO: A high-performance theorem prover. *JAR*, 8, 183–212.
- Levesque**, H. J. e Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3, 78–93.
- Levin**, D. A., Peres, Y. e Wilmer, E. L. (2008). *Markov Chains and Mixing Times*. American Mathematical Society.
- Levine**, S., Finn, C., Darrell, T. e Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *JMLR*, 17, 1334–1373.
- Levine**, S., Pastor, P., Krizhevsky, A., Ibarz, J. e Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, 37, 421–436.
- Levy**, D. (1989). The million pound bridge program. In Levy, D. e Beal, D. (a cura di), *Heuristic Programming in Artificial Intelligence*. Ellis Horwood.
- Levy**, D. (2008). *Love and Sex with Robots: The Evolution of Human—Robot Relationships*. Harper.
- Levy**, O. e Goldberg, Y. (2014). Linguistic regularities in sparse and explicit word representations. In *Proc. Eighteenth Conference on Computational Natural Language Learning*.
- Leyton-Brown**, K. e Shoham, Y. (2008). *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan & Claypool.
- Li**, C. M. e Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *IJCAI-97*.
- Li**, K. e Malik, J. (2018a). Implicit maximum likelihood estimation. arXiv:1809.09087.
- Li**, K. e Malik, J. (2018b). On the implicit assumptions of GANs. arXiv:1811.12402.
- Li**, M., Vitányi, P., et al. (2008). *An Introduction to Kolmogorov Complexity and Its Applications* (3rd edition). Springer-Verlag.
- Li**, T.-M., Gharbi, M., Adams, A., Durand, F. e Ragan-Kelley, J. (2018). Differentiable programming for image processing and deep learning in Halide. *ACM Transactions on Graphics*, 37, 139.
- Li**, W. e Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proc. 1st International Conference on Informatics in Control, Automation and Robotics*.
- Li**, X. e Yao, X. (2012). Cooperatively coevolving particle swarms for large scale optimization. *IEEE Trans. Evolutionary Computation*, 16, 210–224.
- Li**, Z., Li, P., Krishnan, A. e Liu, J. (2011). Large-scale dynamic gene regulatory network inference combining differential equation models with local dynamic Bayesian network analysis. *Bioinformatics*, 27 19, 2686–91.
- Liang**, P., Jordan, M. I. e Klein, D. (2011). Learning dependency-based compositional semantics. arXiv:1109.6841.
- Liang**, P. e Potts, C. (2015). Bringing machine learning and compositional semantics together. *Annual Review of Linguistics*, 1, 355–376.
- Liberatore**, P. (1997). The complexity of the language A. *Electronic Transactions on Artificial Intelligence*, 1, 13–38.
- Lifschitz**, V. (2001). Answer set programming and plan generation. *AII*, 138, 39–54.
- Lighthill**, J. (1973). Artificial intelligence: A general survey. In Lighthill, J., Sutherland, N. S., Needham, R. M., Longuet-Higgins, H. C. e Michie, D. (a cura di), *Artificial Intelligence: A Paper Symposium*. Science Research Council of Great Britain.
- Lillicrap**, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. e Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv:1509.02971.
- Lin**, S. (1965). Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, 44(10), 2245–2269.
- Lin**, S. e Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, 21, 498–516.
- Lindley**, D. V. (1956). On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, 27, 986–1005.
- Lindsay**, R. K., Buchanan, B. G., Feigenbaum, E. A. e Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill.
- Lindsten**, F., Jordan, M. I. e Schön, T. B. (2014). Particle Gibbs with ancestor sampling. *JMLR*, 15, 2145–2184.
- Littman**, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *ICML-94*.
- Littman**, M. L., Cassandra, A. R. e Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *ICML-95*.
- Littman**, M. L. (2015). Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521, 445–451.

- Littman**, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M. e MacGlashan, J. (2017). Environment-independent task specifications via GLTL. arXiv:1704.04341.
- Liu**, B., Gemp, I., Ghavamzadeh, M., Liu, J., Mahadevan, S. e Petrik, M. (2018). Proximal gradient temporal difference learning: Stable reinforcement learning with polynomial sample complexity. *JAIR*, 63, 461–494.
- Liu**, H., Simonyan, K., Vinyals, O., Fernando, C. e Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. arXiv:1711.00436.
- Liu**, H., Simonyan, K. e Yang, Y. (2019). DARTS: Differentiable architecture search. In *ICLR-19*.
- Liu**, J. e Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *JASA*, 93, 1022–1031.
- Liu**, J. e West, M. (2001). Combined parameter and state estimation in simulation-based filtering. In Doucet, A., de Freitas, J. F. G. e Gordon, N. (a cura di), *Sequential Monte Carlo Methods in Practice*. Springer.
- Liu**, L. T., Dean, S., Rolf, E., Simchowitz, M. e Hardt, M. (2018a). Delayed impact of fair machine learning. arXiv:1803.04383.
- Liu**, M.-Y., Breuel, T. e Kautz, J. (2018b). Unsupervised image-to-image translation networks. In *NeurIPS 30*.
- Liu**, X., Faes, L., Kale, A. U., Wagner, S. K., Fu, D. J., Bruynseels, A., Mahendiran, T., Moraes, G., Shamas, M., Kern, C., Ledsam, J. R., Schmid, M., Balaskas, K., Topol, E., Bachmann, L. M., Keane, P. A. e Denniston, A. K. (2019a). A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: A systematic review and meta-analysis. *The Lancet Digital Health*.
- Liu**, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. e Stoyanov, V. (2019b). RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692.
- Liu**, Y., Jain, A., Eng, C., Way, D. H., Lee, K., Bui, P., Kanada, K., de Oliveira Marinho, G., Gallegos, J., Gabriele, S., Gupta, V., Singh, N., Natarajan, V., Hofmann-Wellenhof, R., Corrado, G., Peng, L., Webster, D. R., Ai, D., Huang, S., Liu, Y., Dunn, R. C. e Coz, D. (2019c). A deep learning system for differential diagnosis of skin diseases. arXiv:1909.
- Liu**, Y., Gadepalli, K. K., Norouzi, M., Dahl, G., Kohlberger, T., Venugopalan, S., Boyko, A. S., Timofeev, A., Nelson, P. Q., Corrado, G., Hipp, J. D., Peng, L. e Stumpe, M. C. (2017). Detecting cancer metastases on gigapixel pathology images. arXiv:1703.02442.
- Liu**, Y., Kohlberger, T., Norouzi, M., Dahl, G., Smith, J. L., Mohtashamian, A., Olson, N., Peng, L., Hipp, J. D. e Stumpe, M. C. (2018). Artificial intelligence-based breast cancer nodal metastasis detection: Insights into the black box for pathologists. *Archives of Pathology & Laboratory Medicine*, 143, 859–868.
- Livescu**, K., Glass, J. e Bilmes, J. (2003). Hidden feature modeling for speech recognition using dynamic Bayesian networks. In *EUROSPEECH-2003*.
- Lloyd**, S. (2000). Ultimate physical limits to computation. *Nature*, 406, 1047–1054.
- Lloyd**, W. F. (1833). *Two Lectures on the Checks to Population*. Oxford University.
- Llull**, R. (1305). *Ars Magna*. Published as Salzinger, I. et al. (a cura di), *Raymundi Lulli Opera omnia*, Mainz, 1721–1742.
- Loftus**, E. e Palmer, J. (1974). Reconstruction of automobile destruction: An example of the interaction between language and memory. *J. Verbal Learning and Verbal Behavior*, 13, 585–589.
- Lohn**, J. D., Kraus, W. F. e Colombano, S. P. (2001). Evolutionary optimization of yagi-uda antennas. In *Proc. Fourth International Conference on Evolvable Systems*.
- Longuet-Higgins**, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293, 133–135.
- Loos**, S., Irving, G., Szegedy, C. e Kaliszyk, C. (2017). Deep network guided proof search. In *Proc. 21st Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning*.
- Lopez de Segura**, R. (1561). *Libro de la invencion liberal y arte del juego del axedrez*. Andres de Angulo.
- Lorentz**, R. (2015). Early playout termination in MCTS. In Plaat, A., van den Herik, J. e Kosters, W. (a cura di), *Advances in Computer Games*. Springer-Verlag.
- Love**, N., Hinrichs, T. e Genesereth, M. R. (2006). General game playing: Game description language specification. Tech. rep., Stanford University Computer Science Dept.
- Lovejoy**, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28, 47–66.
- Lovelace**, A. (1843). Sketch of the analytical engine invented by Charles Babbage. Note aggiunte alla traduzione effettuata da Lovelace di un articolo con il titolo citato, scritto da L.F. Menabrea sulla base di lezioni tenute da Charles Babbage nel 1840. La traduzione apparve in R. Taylor (a cura di), *Scientific Memoirs, vol. III*. R. e J. E. Taylor, London.
- Loveland**, D. (1970). A linear format for resolution. In *Proc. IRIA Symposium on Automatic Demonstration*.
- Lowe**, D. (1987). Three-dimensional object recognition from single two-dimensional images. *AIJ*, 31, 355–395.
- Lowe**, D. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, 60, 91–110.

- Löwenheim, L.** (1915). Über möglichkeiten im Relativkalkü l. *Mathematische Annalen*, 76, 447–470.
- Lowerre, B. T.** (1976). *The HARPY Speech Recognition System*. Ph.D. thesis, Computer Science Department, Carnegie-Mellon University.
- Lowry, M.** (2008). Intelligent software engineering tools for NASA's crew exploration vehicle. In *ISMIS-08*.
- Loyd, S.** (1959). *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner*. Dover.
- Lozano-Perez, T.** (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32, 108–120.
- Lozano-Perez, T., Mason, M. e Taylor, R.** (1984). Automatic synthesis of fine-motion strategies for robots. *Int. J. Robotics Research*, 3, 3–24.
- Lu, F. e Milios, E.** (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4, 333–349.
- Lubberts, A. e Miikkulainen, R.** (2001). Co-evolving a Go-playing neural network. In *GECCO-01*.
- Luby, M., Sinclair, A. e Zuckerman, D.** (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47, 173–180.
- Lucas, J. R.** (1961). Minds, machines e Gödel. *Philosophy*, 36.
- Lucas, J. R.** (1976). This Gödel is killing me: A re-joinder. *Philosophia*, 6, 145–148.
- Lucas, P., van der Gaag, L. e Abu-Hanna, A.** (2004). Bayesian networks in biomedicine and healthcare. *Artificial Intelligence in Medicine*.
- Luce, D. R. e Raiffa, H.** (1957). *Games and Decisions*. Wiley.
- Lukasiewicz, T.** (1998). Probabilistic logic programming. In *ECAI-98*.
- Lundberg, S. M. e Lee, S.-I.** (2018). A unified approach to interpreting model predictions. In *NeurIPS 30*.
- Lunn, D., Jackson, C., Best, N., Thomas, A. e Spiegelhalter, D. J.** (2013). *The BUGS Book: A Practical Introduction to Bayesian Analysis*. Chapman and Hall.
- Lunn, D., Thomas, A., Best, N. e Spiegelhalter, D. J.** (2000). WinBUGS—a Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, 10, 325–337.
- Luo, S., Bimbo, J., Dahiya, R. e Liu, H.** (2017). Robotic tactile perception of object properties: A review. *Mechatronics*, 48, 54–67.
- Lyman, P. e Varian, H. R.** (2003). How much information? [www.sims.berkeley.edu/how-much-info-](http://www.sims.berkeley.edu/how-much-info/) 2003.
- Lynch, K. e Park, F. C.** (2017). *Modern Robotics*. Cambridge University Press.
- Machina, M.** (2005). Choice under uncertainty. In *Encyclopedia of Cognitive Science*. Wiley.
- MacKay, D. J. C.** (2002). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.
- MacKenzie, D.** (2004). *Mechanizing Proof*. MIT Press.
- Mackworth, A. K.** (1977). Consistency in networks of relations. *AIJ*, 8, 99–118.
- Mackworth, A. K. e Freuder, E. C.** (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *AIJ*, 25, 65–74.
- Madhavan, R. e Schlenoff, C. I.** (2003). Moving object prediction for off-road autonomous navigation. In *Unmanned Ground Vehicle Technology V*.
- Mailath, G. e Samuelson, L.** (2006). *Repeated Games and Reputations: Long-Run Relationships*. Oxford University Press.
- Majercik, S. M. e Littman, M. L.** (2003). Contingent planning under uncertainty via stochastic satisfiability. *AIJ*, 147, 119–162.
- Malhotra, P., Vig, L., Shroff, G. e Agarwal, P.** (2015). Long short term memory networks for anomaly detection in time series. In *ISANN-15*.
- Malik, D., Palaniappan, M., Fisac, J. F., Hadfield-Menell, D., Russell, S. J. e Dragan, A. D.** (2018). An efficient, generalized bellman update for cooperative inverse reinforcement learning. In *ICML-18*.
- Malone, T. W.** (2004). *The Future of Work*. Harvard Business Review Press.
- Maneva, E., Mossel, E. e Wainwright, M.** (2007). A new look at survey propagation and its generalizations. arXiv:cs/0409012.
- Manna, Z. e Waldinger, R.** (1971). Toward automatic program synthesis. *CACM*, 14, 151–165.
- Manna, Z. e Waldinger, R.** (1985). *The Logical Basis for Computer Programming: Volume 1: Deductive Reasoning*. Addison-Wesley.
- Manne, A. S.** (1960). Linear programming and sequential decisions. *Management Science*, 6, 259–267.
- Manning, C. e Schütze, H.** (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.
- Manning, C., Raghavan, P. e Schütze, H.** (2008). *Introduction to Information Retrieval*. Cambridge University Press.
- Mannion, M.** (2002). Using first-order logic for product line model validation. In *Software Product Lines: Second International Conference*.

- Mansinghka**, V. K., Selsam, D. e Perov, Y. (2013). Venture: A higher-order probabilistic programming platform with programmable inference. arXiv:1404.0099.
- Marbach**, P. e Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. Technical report, Laboratory for Information and Decision Systems, MIT.
- Marcus**, G. (2009). *Kluge: The Haphazard Evolution of the Human Mind*. Mariner Books.
- Marcus**, M. P., Santorini, B. e Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19, 313–330.
- Marinescu**, R. e Dechter, R. (2009). AND/OR branch-and-bound search for combinatorial optimization in graphical models. *AIJ*, 173, 1457–1491.
- Markov**, A. (1913). An example of statistical investigation in the text of “Eugene Onegin” illustrating coupling of “tests” in chains. *Proc. Academy of Sciences of St. Petersburg*, 7, 153–162.
- Marler**, R. T. e Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26, 369–395.
- Maron**, M. E. (1961). Automatic indexing: An experimental inquiry. *JACM*, 8, 404–417.
- Márquez**, L. e Rodríguez, H. (1998). Part-of-speech tagging using decision trees. In *ECML-98*.
- Marr**, D. e Poggio, T. (1976). Cooperative computation of stereo disparity. *Science*, 194, 283–287.
- Marr**, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman.
- Marriott**, K. e Stuckey, P. J. (1998). *Programming with Constraints: An Introduction*. MIT Press.
- Marsland**, S. (2014). *Machine Learning: An Algorithmic Perspective* (2nd edition). CRC Press.
- Martelli**, A. e Montanari, U. (1973). Additive AND/OR graphs. In *IJCAI-73*.
- Martelli**, A. (1977). On the complexity of admissible search algorithms. *AIJ*, 8, 1–13.
- Marthi**, B., Pasula, H., Russell, S. J. e Peres, Y. (2002). Decayed MCMC filtering. In *UAI-02*.
- Marthi**, B., Russell, S. J., Latham, D. e Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. In *IJCAI-05*.
- Marthi**, B., Russell, S. J. e Wolfe, J. (2007). Angelic semantics for high-level actions. In *ICAPS-07*.
- Marthi**, B., Russell, S. J. e Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS-08*.
- Martin**, D., Fowlkes, C. e Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *PAMI*, 26, 530–549.
- Martin**, F. G. (2012). Will massive open online courses change how we teach? *CACM*, 55, 26–28.
- Martin**, J. H. (1990). *A Computational Model of Metaphor Interpretation*. Academic Press.
- Mason**, M. (1993). Kicking the sensing habit. *AIMag*, 14, 58–59.
- Mason**, M. (2001). *Mechanics of Robotic Manipulation*. MIT Press.
- Mason**, M. e Salisbury, J. (1985). *Robot Hands and the Mechanics of Manipulation*. MIT Press.
- Mataric**, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4, 73–83.
- Mates**, B. (1953). *Stoic Logic*. University of California Press.
- Matuszek**, C., Cabral, J., Witbrock, M. e DeOliveira, J. (2006). An introduction to the syntax and semantics of Cyc. In *Proc. AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*.
- Mausam** and Kolobov, A. (2012). *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool.
- Maxwell**, J. (1868). On governors. *Proc. Roy. Soc.*, 16, 270–283.
- Mayer**, J., Khairy, K. e Howard, J. (2010). Drawing an elephant with four complex parameters. *American Journal of Physics*, 78, 648–649.
- Mayor**, A. (2018). *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology*. Princeton University Press.
- McAllester**, D. A. (1980). An outlook on truth maintenance. AI memo, MIT AI Laboratory.
- McAllester**, D. A. (1988). Conspiracy numbers for min-max search. *AIJ*, 35, 287–310.
- McAllester**, D. A. (1998). What is the most pressing issue facing AI and the AAAI today? Candidate statement, election for Councilor of the American Association for Artificial Intelligence.
- McAllester**, D. A. e Rosenblitt, D. (1991). Systematic nonlinear planning. In *AAAI-91*.
- McAllester**, D. A. (1990). Truth maintenance. In *AAAI-90*.
- McAllester**, D. A., Milch, B. e Goodman, N. D. (2008). Random-world semantics and syntactic independence for expressive languages. Technical report, MIT.
- McCallum**, A. (2003). Efficiently inducing features of conditional random fields. In *UAI-03*.

- McCallum**, A., Schultz, K. e Singh, S. (2009). FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *NeurIPS 22*.
- McCarthy**, J. (1958). Programs with common sense. In *Proc. Symposium on Mechanisation of Thought Processes*.
- McCarthy**, J. (1963). Situations, actions, and causal laws. Memo, Stanford University Artificial Intelligence Project.
- McCarthy**, J. (1968). Programs with common sense. In Minsky, M. L. (a cura di), *Semantic Information Processing*. MIT Press.
- McCarthy**, J. (1980). Circumscription: A form of non-monotonic reasoning. *AIJ*, 13, 27–39.
- McCarthy**, J. (2007). From here to human-level AI. *AIJ*, 171.
- McCarthy**, J. e Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D. e Swann, M. (a cura di), *Machine Intelligence 4*. Edinburgh University Press.
- McCawley**, J. D. (1988). *The Syntactic Phenomena of English*. University of Chicago Press.
- McCorduck**, P. (2004). *Machines Who Think: A Personal Inquiry Into the History and Prospects of Artificial Intelligence* (Revised edition). A K Peters.
- McCulloch**, W. S. e Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115–137.
- McCune**, W. (1997). Solution of the Robbins problem. *JAR*, 19, 263–276.
- McCune**, W. (1990). Otter 2.0. In *International Conference on Automated Deduction*.
- McDermott**, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*, 57, 4–9.
- McDermott**, D. (1978a). Planning and acting. *Cognitive Science*, 2, 71–109.
- McDermott**, D. (1978b). Tarskian semantics, or no notation without denotation! *Cognitive Science*, 2, 277–282.
- McDermott**, D. (1985). Reasoning about plans. In Hobbs, J. e Moore, R. (a cura di), *Formal theories of the commonsense world*. Ablex.
- McDermott**, D. (1987). A critique of pure reason. *Computational Intelligence*, 3, 151–237.
- McDermott**, D. (1996). A heuristic estimator for means-ends analysis in planning. In *ICAPS-96*.
- McDermott**, D. e Doyle, J. (1980). Non-monotonic logic: i. *AIJ*, 13, 41–72.
- McDermott**, J. (1982). R1: A rule-based configurer of computer systems. *AIJ*, 19, 39–88.
- McEliece**, R. J., MacKay, D. J. C. e Cheng, J.-F. (1998). Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16, 140–152.
- McGregor**, J. J. (1979). Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19, 229–250.
- McIlraith**, S. e Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, 16, 46–53.
- McKinney**, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas*. O'Reilly.
- McLachlan**, G. J. e Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley.
- McMahan**, H. B. e Andrew, G. (2018). A general approach to adding differential privacy to iterative training procedures. arXiv:1812.06210.
- McMillan**, K. L. (1993). *Symbolic Model Checking*. Kluwer.
- McWhorter**, J. H. (2014). *The Language Hoax: Why the World Looks the Same in Any Language*. Oxford University Press.
- Meehl**, P. (1955). *Clinical vs. Statistical Prediction*. University of Minnesota Press.
- Mehrabi**, N., Morstatter, F., Saxena, N., Lerman, K. e Galstyan, A. (2019). A survey on bias and fairness in machine learning. arXiv:1908.09635.
- Mendel**, G. (1866). Versuche über pflanzen-hybriden. *Verhandlungen des Naturforschenden Vereins, Abhandlungen*, Brünn, 4, 3–47. Tradotto in inglese da C. T. Druery, published by Bateson (1902).
- Mercer**, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Phil. Trans. Roy. Soc., A*, 209, 415–446.
- Merleau-Ponty**, M. (1945). *Phenomenology of Perception*. Routledge.
- Metropolis**, N., Rosenbluth, A., Rosenbluth, M., Teller, A. e Teller, E. (1953). Equations of state calculations by fast computing machines. *J. Chemical Physics*, 21, 1087–1091.
- Metropolis**, N. e Ulam, S. (1949). The beginning of the Monte Carlo method. *Journal of the American Statistical Association*, 44, 335–341.
- Mézard**, M., Parisi, G. e Virasoro, M. (1987). *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*. World Scientific.
- Michie**, D. (1966). Game-playing and game-learning automata. In Fox, L. (a cura di), *Advances in Programming and Non-Numerical Computation*. Pergamon.
- Michie**, D. (1972). Machine intelligence at Edinburgh. *Management Informatics*, 2, 7–12.

- Michie**, D. e Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. e Michie, (a cura di), *Machine Intelligence 2*. Elsevier.
- Michie**, D. (1963). Experiments on the mechanization of game-learning Part I. Characterization of the model and its parameters. *The Computer Journal*, 6, 232–236.
- Miikkulainen**, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier.
- Mikolov**, T., Chen, K., Corrado, G. e Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781.
- Mikolov**, T., Karafiat, M., Burget, L., Černocký, J. e Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.
- Mikolov**, T., Sutskever, I., Chen, K., Corrado, G. e Dean, J. (2014). Distributed representations of words and phrases and their compositionality. In *NeurIPS 26*.
- Milch**, B. (2006). *Probabilistic Models with Unknown Objects*. Ph.D. thesis, UC Berkeley.
- Milch**, B., Marthi, B., Sontag, D., Russell, S. J., Ong, D. e Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *IJCAI-05*.
- Milch**, B., Zettlemoyer, L., Kersting, K., Haimes, M. e Kaelbling, L. P. (2008). Lifted probabilistic inference with counting formulas. In *AAAI-08*.
- Milgrom**, P. (1997). Putting auction theory to work: The simultaneous ascending auction. Tech. rep., Stanford University Department of Economics.
- Mill**, J. S. (1863). *Utilitarianism*. Parker, Son and Bourn, London.
- Miller**, A. C., Merkhofer, M. M., Howard, R. A., Matheson, J. E. e Rice, T. R. (1976). Development of automated aids for decision analysis. Technical report, SRI International.
- Miller**, T., Howe, P. e Sonenberg, L. (2017). Explainable AI: Beware of inmates running the asylum. In *Proc. IJCAI-17 Workshop on Explainable AI*.
- Minka**, T. (2010). Bayesian linear regression. Manoscritto non pubblicato.
- Minka**, T., Cleven, R. e Zaykov, Y. (2018). TrueSkill 2: An improved Bayesian skill rating system. Tech. rep., Microsoft Research.
- Minker**, J. (2001). *Logic-Based Artificial Intelligence*. Kluwer.
- Minsky**, M. L. (1975). A framework for representing knowledge. In Winston, P. H. (a cura di), *The Psychology of Computer Vision*. McGraw-Hill.
- Minsky**, M. L. (1986). *The Society of Mind*. Simon and Schuster.
- Minsky**, M. L. (2007). *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon and Schuster.
- Minsky**, M. L. e Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.
- Minsky**, M. L. e Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry* (Expanded edition). MIT Press.
- Minsky**, M. L., Singh, P. e Sloman, A. (2004). The St. Thomas common sense symposium: Designing architectures for human-level intelligence. *AIMag*, 25, 113–124.
- Minton**, S., Johnston, M. D., Philips, A. B. e Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *AII*, 58, 161–205.
- Mirjalili**, S. M. e Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69, 46–61.
- Misak**, C. (2004). *The Cambridge Companion to Peirce*. Cambridge University Press.
- Mitchell**, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. e Gebru, T. (2019). Model cards for model reporting. *Proc. of the Conference on Fairness, Accountability, and Transparency*.
- Mitchell**, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.
- Mitchell**, M. (2019). *Artificial Intelligence: A Guide for Thinking Humans*. Farrar, Straus and Giroux.
- Mitchell**, M., Holland, J. H. e Forrest, S. (1996). When will a genetic algorithm outperform hill climbing? In *NeurIPS 6*.
- Mitchell**, T. M. (1997). *Machine Learning*. McGraw-Hill.
- Mitchell**, T. M. (2005). Reading the web: A breakthrough goal for AI. *AIMag*, 26.
- Mitchell**, T. M. (2007). Learning, information extraction and the web. In *ECML-07*.
- Mitchell**, T. M., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al. (2018). Never-ending learning. *CACM*, 61, 103–115.
- Mitchell**, T. M., Shinkareva, S. V., Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. A. e Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. *Science*, 320, 1191–1195.

- Mittelstadt, B.** (2019). Principles alone cannot guarantee ethical AI. *Nature Machine Intelligence*, 1, 501–507.
- Mitten, L. G.** (1960). An analytic solution to the least cost testing sequence problem. *Journal of Industrial Engineering*, 11, 17.
- Miyato, T.**, Kataoka, T., Koyama, M. e Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. arXiv:1802.05957.
- Mnih, V.**, Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. e Riedmiller, M. A. (2013). Playing Atari with deep reinforcement learning. arXiv:1312.5602.
- Mnih, V.**, Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. e Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.
- Mohr, R.** e Henderson, T. C. (1986). Arc and path consistency revisited. *AIJ*, 28, 225–233.
- Montague, R.** (1970). English as a formal language. In Visentini, B. (a cura di), *Linguaggi nella Società e nella Tecnica*. Edizioni di Comunità.
- Montague, R.** (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsik, J. M. E. e Suppes, P. (a cura di), *Approaches to Natural Language*. D. Reidel.
- Montanari, U.** (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95–132.
- Montemerlo, M.** e Thrun, S. (2004). Large-scale robotic 3-D mapping of urban structures. In *Proc. International Symposium on Experimental Robotics*.
- Montemerlo, M.**, Thrun, S., Koller, D. e Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI-02*.
- Mooney, R.** (1999). Learning for semantic interpretation: Scaling up without dumbing down. In *Proc. 1st Workshop on Learning Language in Logic*.
- Moore, A. M.** e Wong, W.-K. (2003). Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *ICML-03*.
- Moore, A. W.** e Atkeson, C. G. (1993). Prioritized sweeping—Reinforcement learning with less data and less time. *Machine Learning*, 13, 103–130.
- Moore, A. W.** e Lee, M. S. (1997). Cached sufficient statistics for efficient machine learning with large datasets. *JAIR*, 8, 67–91.
- Moore, E. F.** (1959). The shortest path through a maze. In *Proc. International Symposium on the Theory of Switching, Part II*. Harvard University Press.
- Moore, R. C.** (1980). Reasoning about knowledge and action. Artificial intelligence center technical note, SRI International.
- Moore, R. C.** (1985). A formal theory of knowledge and action. In Hobbs, J. R. e Moore, R. C. (a cura di), *Formal Theories of the Commonsense World*. Ablex.
- Moore, R. C.** e DeNero, J. (2011). L1 and L2 regularization for multiclass hinge loss models. In *Symposium on Machine Learning in Speech and Natural Language Processing*.
- Moravčík, M.**, Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M. e Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in no-limit poker. arXiv:1701.01724.
- Moravec, H. P.** (1983). The Stanford cart and the CMU rover. *Proc. IEEE*, 71, 872–884.
- Moravec, H. P.** e Elfes, A. (1985). High resolution maps from wide angle sonar. In *ICRA-85*.
- Moravec, H. P.** (2000). *Robot: Mere Machine to Transcendent Mind*. Oxford University Press.
- Morgan, C. L.** (1896). *Habit and Instinct*. Edward Arnold.
- Morgan, T. J. H.** e Griffiths, T. L. (2015). What the Baldwin Effect affects. In *COGSCI-15*.
- Morjaria, M. A.**, Rink, F. J., Smith, W. D., Klempner, G., Burns, C. e Stein, J. (1995). Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *UAI-95*.
- Morrison, P.** e Morrison, E. (a cura di). (1961). *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others*. Dover.
- Moskewicz, M. W.**, Madigan, C. F., Zhao, Y., Zhang, L. e Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proc. 38th Design Automation Conference*.
- Mott, A.**, Job, J., Vlimant, J.-R., Lidar, D. e Spiropulu, M. (2017). Solving a Higgs optimization problem with quantum annealing for machine learning. *Nature*, 550, 375.
- Motzkin, T. S.** e Schoenberg, I. J. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6, 393–404.
- Moutarlier, P.** e Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environment modeling. In *ISRR-89*.

- Mueller**, E. T. (2006). *Commonsense Reasoning*. Morgan Kaufmann.
- Muggleton**, S. H. e De Raedt, L. (1994). Inductive logic programming: Theory and methods. *J. Logic Programming*, 19/20, 629–679.
- Müller**, M. (2002). Computer Go. *AIJ*, 134, 145–179.
- Mumford**, D. e Shah, J. (1989). Optimal approximations by piece-wise smooth functions and associated variational problems. *Commun. Pure Appl. Math.*, 42, 577–685.
- Mundy**, J. e Zisserman, A. (a cura di). (1992). *Geometric Invariance in Computer Vision*. MIT Press.
- Munos**, R., Stepleton, T., Harutyunyan, A. e Bellemare, M. G. (2017). Safe and efficient off-policy reinforcement learning. In *NeurIPS* 29.
- Murphy**, K. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, UC Berkeley.
- Murphy**, K. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
- Murphy**, K. e Mian, I. S. (1999). Modelling gene expression data using Bayesian networks. Tech. rep., Computer Science Division, UC Berkeley.
- Murphy**, K. e Russell, S. J. (2001). Rao-Blackwellised particle filtering for dynamic Bayesian networks. In Doucet, A., de Freitas, J. F. G. e Gordon, N. J. (a cura di), *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.
- Murphy**, K. e Weiss, Y. (2001). The factored frontier algorithm for approximate inference in DBNs. In *UAI-01*.
- Murphy**, R. (2000). *Introduction to AI Robotics*. MIT Press.
- Murray**, L. M. (2013). Bayesian state-space modelling on high-performance hardware using LibBi. arXiv:1306.3277.
- Murray**, R. M. (2017). *A Mathematical Introduction to Robotic Manipulation*. CRC Press.
- Murray-Rust**, P., Rzepa, H. S., Williamson, J. e Willighagen, E. L. (2003). Chemical markup, XML and the world-wide web. 4. CML schema. *J. Chem. Inf. Comput. Sci.*, 43, 752–772.
- Murthy**, C. e Russell, J. R. (1990). A constructive proof of Higman's lemma. In *LICS-90*.
- Muscettola**, N. (2002). Computing the envelope for stepwise-constant resource allocations. In *CP-02*.
- Muscettola**, N., Nayak, P., Pell, B. e Williams, B. (1998). Remote agent: To boldly go where no AI system has gone before. *AIJ*, 103, 5–48.
- Muslea**, I. (1999). Extraction patterns for information extraction tasks: A survey. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*.
- Muth**, J. T., Vogt, D. M., Truby, R. L., Mengüç, Y., Kolesky, D. B., Wood, R. J. e Lewis, J. A. (2014). Embedded 3D printing of strain sensors within highly stretchable elastomers. *Advanced Materials*, 26, 6307–6312.
- Myerson**, R. (1981). Optimal auction design. *Mathematics of Operations Research*, 6, 58–73.
- Myerson**, R. (1986). Multistage games with communication. *Econometrica*, 54, 323–358.
- Myerson**, R. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press.
- Nair**, V. e Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML-10*.
- Nalwa**, V. S. (1993). *A Guided Tour of Computer Vision*. Addison-Wesley.
- Narayanan**, A., Shi, E. e Rubinstein, B. I. (2011). Link prediction by de-anonymization: How we won the Kaggle social network challenge. In *IJCNN-11*.
- Narayanan**, A. e Shmatikov, V. (2006). How to break anonymity of the Netflix prize dataset. arXiv:cs/0610105.
- Nash**, J. (1950). Equilibrium points in N-person games. *PNAS*, 36, 48–49.
- Nash**, P. (1973). *Optimal Allocation of Resources Between Research Projects*. Ph.D. thesis, University of Cambridge.
- Nayak**, P. e Williams, B. (1997). Fast context switching in real-time propositional reasoning. In *AAAI-97*.
- Neches**, R., Swartout, W. R. e Moore, J. D. (1985). Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11, 1337–1351.
- Nemhauser**, G. L., Wolsey, L. A. e Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming*, 14, 265–294.
- Nesterov**, Y. e Nemirovski, A. (1994). *Interior-Point Polynomial Methods in Convex Programming*. SIAM (Society for Industrial and Applied Mathematics).
- Newell**, A. (1982). The knowledge level. *AIJ*, 18, 82–127.
- Newell**, A. (1990). *Unified Theories of Cognition*. Harvard University Press.
- Newell**, A. e Ernst, G. (1965). The search for generality. In *Proc. IFIP Congress*.

- Newell**, A., Shaw, J. C. e Simon, H. A. (1957). Empirical explorations with the logic theory machine. *Proc. Western Joint Computer Conference*, 15, 218–239. Reprinted in Feigenbaum and Feldman (1963).
- Newell**, A. e Simon, H. A. (1961). GPS, a program that simulates human thought. In Billing, H. (a cura di), *Lernende Automaten*. R. Oldenbourg.
- Newell**, A. e Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall.
- Newell**, A. e Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *CACM*, 19, 113–126.
- Newton**, I. (1664–1671). *Methodus fluxionum et serierum infinitarum*. Note non pubblicate.
- Ng**, A. Y. (2004). Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *ICML-04*.
- Ng**, A. Y. (2019). *Machine Learning Yearning*. www.mlyearning.org.
- Ng**, A. Y., Harada, D. e Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML-99*.
- Ng**, A. Y. e Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI-00*.
- Ng**, A. Y. e Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *NeurIPS 14*.
- Ng**, A. Y., Kim, H. J., Jordan, M. I. e Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. In *NeurIPS 16*.
- Ng**, A. Y. e Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *ICML-00*.
- Nicholson**, A. e Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI-92*.
- Nielsen**, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
- Nielsen**, T. e Jensen, F. (2003). Sensitivity analysis in influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 33, 223–234.
- Niemelä**, I., Simons, P. e Syrjänen, T. (2000). Smodels: A system for answer set programming. In *Proc. 8th International Workshop on Non-Monotonic Reasoning*.
- Nikolaidis**, S. e Shah, J. (2013). Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *HRI-13*.
- Niles**, I. e Pease, A. (2001). Towards a standard upper ontology. In *Proc. International Conference on Formal Ontology in Information Systems*.
- Nilsson**, D. e Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *UAI-00*.
- Nilsson**, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill.
- Nilsson**, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.
- Nilsson**, N. J. (1984). Shakey the robot. Technical note, SRI International.
- Nilsson**, N. J. (1986). Probabilistic logic. *AIJ*, 28, 71–87.
- Nilsson**, N. J. (1995). Eye on the prize. *AIMag*, 16, 9–17.
- Nilsson**, N. J. (2009). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press.
- Nisan**, N. (2007). Introduction to mechanism design (for computer scientists). In Nisan, N., Roughgarden, T., Tardos, E. e Vazirani, V. V. (a cura di), *Algorithmic Game Theory*. Cambridge University Press.
- Nisan**, N., Roughgarden, T., Tardos, E. e Vazirani, V. (a cura di). (2007). *Algorithmic Game Theory*. Cambridge University Press.
- Niv**, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53, 139–154.
- Nivre**, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C., McDonald, R., Petrov, S., et al. (2016). Universal dependencies v1: A multilingual treebank collection. In *Proc. International Conference on Language Resources and Evaluation*.
- Nodelman**, U., Shelton, C. e Koller, D. (2002). Continuous time Bayesian networks. In *UAI-02*.
- Noe**, A. (2009). *Out of Our Heads: Why You Are Not Your Brain, and Other Lessons from the Biology of Consciousness*. Hill and Wang.
- Nordfors**, D., Cerf, V. e Senges, M. (2018). *Disrupting Unemployment*. Amazon Digital Services.
- Norvig**, P. (1988). Multiple simultaneous interpretations of ambiguous sentences. In *COGSCI-88*.
- Norvig**, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann.
- Norvig**, P. (2009). Natural language corpus data. In Segaran, T. e Hammerbacher, J. (a cura di), *Beautiful Data*. O'Reilly.
- Nowick**, S. M., Dean, M. E., Dill, D. L. e Horowitz, M. (1993). The design of a high-performance cache controller: A case study in asynchronous synthesis. *Integration: The VLSI Journal*, 15, 241–262.

- Och**, F. J. e Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29, 19–51.
- Och**, F. J. e Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, 30, 417–449.
- Och**, F. J. e Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *COLING-02*.
- Ogawa**, S., Lee, T.-M., Kay, A. R. e Tank, D. W. (1990). Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *PNAS*, 87, 9868–9872.
- Oh**, M.-S. e Berger, J. O. (1992). Adaptive importance sampling in Monte Carlo integration. *Journal of Statistical Computation and Simulation*, 41, 143–168.
- Oh**, S., Russell, S. J. e Sastry, S. (2009). Markov chain Monte Carlo data association for multi-target tracking. *IEEE Transactions on Automatic Control*, 54, 481–497.
- Oizumi**, M., Albantakis, L. e Tononi, G. (2014). From the phenomenology to the mechanisms of consciousness: Integrated information theory 3.0. *PLoS Computational Biology*, 10, e1003588.
- Olesen**, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *PAMI*, 15, 275–279.
- Oliver**, N., Garg, A. e Horvitz, E. J. (2004). Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, 96, 163–180.
- Oliver**, R. M. e Smith, J. Q. (a cura di). (1990). *Influence Diagrams, Belief Nets and Decision Analysis*. Wiley.
- Omohundro**, S. (2008). The basic AI drives. In *AGI-08 Workshop on the Sociocultural, Ethical and Futurological Implications of Artificial Intelligence*.
- O’Neil**, C. (2017). *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Broadway Books.
- O’Neil**, C. e Schutt, R. (2013). *Doing Data Science: Straight Talk from the Frontline*. O’Reilly.
- O'Reilly**, U.-M. e Oppacher, F. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *Proc. Third Conference on Parallel Problem Solving from Nature*.
- Osborne**, M. J. (2004). *An Introduction to Game Theory*. Oxford University Pres.
- Osborne**, M. J. e Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.
- Osherson**, D. N., Stob, M. e Weinstein, S. (1986). *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press.
- Ostrom**, E. (1990). *Governing the Commons*. Cambridge University Press.
- Padgham**, L. e Winikoff, M. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. Wiley.
- Paige**, B. e Wood, F. (2014). A compilation target for probabilistic programming languages. In *ICML-14*.
- Paige**, B., Wood, F., Doucet, A. e Teh, Y. W. (2015). Asynchronous anytime sequential Monte Carlo. In *NeurIPS 27*.
- Palacios**, H. e Geffner, H. (2007). From conformant into classical planning: Efficient translations that may be complete too. In *ICAPS-07*.
- Palmer**, S. (1999). *Vision Science: Photons to Phenomenology*. MIT Press.
- Papadimitriou**, C. H. (1994). *Computational Complexity*. Addison-Wesley.
- Papadimitriou**, C. H. e Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12, 441–450.
- Papadimitriou**, C. H. e Yannakakis, M. (1991). Shortest paths without a map. *Theoretical Computer Science*, 84, 127–150.
- Papavassiliou**, V. e Russell, S. J. (1999). Convergence of reinforcement learning with general function approximators. In *IJCAI-99*.
- Parisi**, G. (1988). *Statistical Field Theory*. Addison-Wesley.
- Parisi**, M. M. G. e Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, 297, 812–815.
- Park**, J. D. e Darwiche, A. (2004). Complexity results and approximation strategies for MAP explanations. *JAIR*, 21, 101–133.
- Parker**, A., Nau, D. S. e Subrahmanian, V. S. (2005). Game-tree search with combinatorially large belief states. In *IJCAI-05*.
- Parker**, D. B. (1985). Learning logic. Technical report, Center for Computational Research in Economics and Management Science, MIT.
- Parker**, L. E. (1996). On the design of behavior-based multi-robot teams. *J. Advanced Robotics*, 10, 547–578.
- Parr**, R. e Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *NeurIPS 10*.

- Parzen, E.** (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33, 1065–1076.
- Pasca, M.** e Harabagiu, S. M. (2001). High performance question/answering. In *SIGIR-01*.
- Pasca, M.**, Lin, D., Bigham, J., Lifchits, A. e Jain, A. (2006). Organizing and searching the world wide web of facts—Step one: The one-million fact extraction challenge. In *AAAI-06*.
- Paskin, M.** (2002). Maximum entropy probabilistic logic. Tech. report, UC Berkeley.
- Pasula, H.**, Marthi, B., Milch, B., Russell, S. J. e Shpitser, I. (2003). Identity uncertainty and citation matching. In *NeurIPS 15*.
- Pasula, H.**, Russell, S. J., Ostland, M. e Ritov, Y. (1999). Tracking many objects with many sensors. In *IJCAI-99*.
- Patel-Schneider, P.** (2014). Analyzing schema.org. In *Proc. International Semantic Web Conference*.
- Patrick, B. G.**, Almulla, M. e Newborn, M. (1992). An upper bound on the time complexity of iterative-deepening-A*. *AIJ*, 5, 265–278.
- Paul, R. P.** (1981). *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press.
- Pauls, A.** e Klein, D. (2009). K-best A* parsing. In *ACL-09*.
- Peano, G.** (1889). *Arithmetices principia, nova methodo exposita*. Fratres Bocca, Torino.
- Pearce, J.**, Tambe, M. e Maheswaran, R. (2008). Solving multiagent networks using distributed constraint optimization. *AIMag*, 29, 47–62.
- Pearl, J.** (1982a). Reverend Bayes on inference engines: A distributed hierarchical approach. In *AAAI-82*.
- Pearl, J.** (1982b). The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *CACM*, 25, 559–564.
- Pearl, J.** (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Pearl, J.** (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. In *COGSCI-85*.
- Pearl, J.** (1986). Fusion, propagation, and structuring in belief networks. *AIJ*, 29, 241–288.
- Pearl, J.** (1987). Evidential reasoning using stochastic simulation of causal models. *AIJ*, 32, 247–257.
- Pearl, J.** (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.
- Pearl, J.** (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.
- Pearl, J.** e McKenzie, D. (2018). *The Book of Why*. Basic Books.
- Pearl, J.** e Verma, T. (1991). A theory of inferred causation. In *KR-91*.
- Pearson, K.** (1895). Contributions to the mathematical theory of evolution, II: Skew variation in homogeneous material. *Phil. Trans. Roy. Soc.*, 186, 343–414.
- Pearson, K.** (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2, 559–572.
- Pease, A.** e Niles, I. (2002). IEEE standard upper ontology: A progress report. *Knowledge Engineering Review*, 17, 65–70.
- Pednault, E. P. D.** (1986). Formulating multiagent, dynamic-world problems in the classical planning framework. In *Reasoning About Actions and Plans: Proc. 1986 Workshop*.
- Pedregosa, F.**, Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *JMLR*, 12, 2825–2830.
- Peirce, C. S.** (1870). Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American Academy of Arts and Sciences*, 9, 317–378.
- Peirce, C. S.** (1883). A theory of probable inference. Note B. The logic of relatives. In Peirce, C. S. (a cura di), *Studies in Logic*, Little, Brown.
- Peirce, C. S.** (1909). Existential graphs. Unpublished manuscript; reprinted in (Buchler 1955).
- Peleg, B.** e Sudholter, P. (2002). *Introduction to the Theory of Cooperative Games* (2nd edition). Springer-Verlag.
- Pelikan, M.**, Goldberg, D. E. e Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *GECCO-99*.
- Pemberton, J. C.** e Korf, R. E. (1992). Incremental planning on graphs with cycles. In *AIPS-92*.
- Penberthy, J. S.** e Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *KR-92*.
- Peng, J.** e Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 2, 437–454.
- Pennington, J.**, Socher, R. e Manning, C. (2014). Glove: Global vectors for word representation. In *EMNLP-14*.
- Penrose, R.** (1989). *The Emperor's New Mind*. Oxford University Press.
- Penrose, R.** (1994). *Shadows of the Mind*. Oxford University Press.
- Peot, M.** e Smith, D. E. (1992). Conditional nonlinear planning. In *ICAPS-92*.
- Pereira, F.** e Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. In *ACL-92*.

- Pereira**, F. e Warren, D. H. D. (1980). Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *AIJ*, 13, 231–278.
- Peters**, J. e Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21, 682–697.
- Peters**, J., Janzing, D. e Schölkopf, B. (2017). *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT press.
- Peters**, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. e Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv:1802.05365.
- Peterson**, C. e Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1, 995–1019.
- Petosa**, N. e Balch, T. (2019). Multiplayer AlphaZero. arXiv:1910.13012.
- Pfeffer**, A. (2001). IBAL: A probabilistic rational programming language. In *IJCAI-01*.
- Pfeffer**, A., Koller, D., Milch, B. e Takusagawa, K. T. (1999). SPOOK: A system for probabilistic object-oriented knowledge representation. In *UAI-99*.
- Pfeffer**, A. (2016). *Practical Probabilistic Programming*. Manning.
- Pfeffer**, A. (2000). *Probabilistic Reasoning for Complex Systems*. Ph.D. thesis, Stanford University.
- Pfeffer**, A. (2007). The design and implementation of IBAL: A general-purpose probabilistic language. In Getoor, L. e Taskar, B. (a cura di), *Introduction to Statistical Relational Learning*. MIT Press.
- Pfeifer**, R., Bongard, J., Brooks, R. A. e Iwasawa, S. (2006). *How the Body Shapes the Way We Think: A New View of Intelligence*. Bradford.
- Pham**, H., Guan, M. Y., Zoph, B., Le, Q. V. e Dean, J. (2018). Efficient neural architecture search via parameter sharing. arXiv:1802.03268.
- Pineau**, J., Gordon, G. e Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI-03*.
- Pinedo**, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Verlag.
- Pinkas**, G. e Dechter, R. (1995). Improving connectionist energy minimization. *JAIR*, 3, 223–248.
- Pinker**, S. (1995). Language acquisition. In Gleitman, L. R., Liberman, M. e Osherson, D. N. (a cura di), *An Invitation to Cognitive Science* (2nd edition). MIT Press.
- Pinker**, S. (2003). *The Blank Slate: The Modern Denial of Human Nature*. Penguin.
- Pinto**, D., McCallum, A., Wei, X. e Croft, W. B. (2003). Table extraction using conditional random fields. In *SIGIR-03*.
- Pinto**, L. e Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA-16*.
- Platt**, J. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Learning*. MIT Press.
- Plotkin**, G. (1972). Building-in equational theories. In Meltzer, B. e Michie, D. (a cura di), *Machine Intelligence* 7. Edinburgh University Press.
- Plummer**, M. (2003). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proc. Third Int'l Workshop on Distributed Statistical Computing*.
- Pnueli**, A. (1977). The temporal logic of programs. In *FOCS-77*.
- Pohl**, I. (1971). Bi-directional search. In Meltzer, B. e Michie, D. (a cura di), *Machine Intelligence* 6. Edinburgh University Press.
- Pohl**, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI-73*.
- Pohl**, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. e Michie, D. (a cura di), *Machine Intelligence* 8. Ellis Horwood.
- Pohl**, I. (1970). Heuristic search viewed as path finding in a graph. *AIJ*, 1, 193–204.
- Poli**, R., Langdon, W. e McPhee, N. (2008). *A Field Guide to Genetic Programming*. Lulu.com.
- Pomerleau**, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer.
- Poole**, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J. e Ganguli, S. (2017). Exponential expressivity in deep neural networks through transient chaos. In *NeurIPS 29*.
- Poole**, D. (1993). Probabilistic Horn abduction and Bayesian networks. *AIJ*, 64, 81–129.
- Poole**, D. (2003). First-order probabilistic inference. In *IJCAI-03*.
- Poole**, D. e Mackworth, A. K. (2017). *Artificial Intelligence: Foundations of Computational Agents* (2 edition). Cambridge University Press.
- Poppe**, R. (2010). A survey on vision-based human action recognition. *Image Vision Comput.*, 28, 976–990.
- Popper**, K. R. (1959). *The Logic of Scientific Discovery*. Basic Books.

- Popper, K. R.** (1962). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books.
- Portner, P. e Partee, B. H.** (2002). *Formal Semantics: The Essential Readings*. Wiley-Blackwell.
- Post, E. L.** (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43, 163–185.
- Poulton, C. e Watts, M.** (2016). MIT and DARPA pack Lidar sensor onto single chip. *IEEE Spectrum*, August 4.
- Poundstone, W.** (1993). *Prisoner's Dilemma*. Anchor.
- Pourret, O., Naïm, P. e Marcot, B.** (2008). *Bayesian Networks: A Practical Guide to Applications*. Wiley.
- Pradhan, M., Provan, G. M., Middleton, B. e Henrion, M.** (1994). Knowledge engineering for large belief networks. In *UAI-94*.
- Prawitz, D.** (1960). An improved proof procedure. *Theoria*, 26, 102–139.
- Press, W. H., Teukolsky, S. A., Vetterling, W. T. e Flannery, B. P.** (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd edition). Cambridge University Press.
- Preston, J. e Bishop, M.** (2002). *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*. Oxford University Press.
- Prieditis, A. E.** (1993). Machine discovery of effective admissible heuristics. *Machine Learning*, 12, 117–141.
- Prosser, P.** (1993). Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9, 268–299.
- Pullum, G. K.** (1991). *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language)*. University of Chicago Press.
- Pullum, G. K.** (1996). Learnability, hyperlearning, and the poverty of the stimulus. In *22nd Annual Meeting of the Berkeley Linguistics Society*.
- Puterman, M. L.** (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.
- Puterman, M. L. e Shin, M. C.** (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24, 1127–1137.
- Putnam, H.** (1963). 'Degree of confirmation' and inductive logic. In Schilpp, P. A. (a cura di), *The Philosophy of Rudolf Carnap*. Open Court.
- Quillian, M. R.** (1961). A design for an understanding machine. Articolo presentato a un convegno: Semantic Problems in Natural Language, King's College, Cambridge, England.
- Quine, W. V.** (1953). Two dogmas of empiricism. In *From a Logical Point of View*. Harper and Row.
- Quine, W. V.** (1960). *Word and Object*. MIT Press.
- Quine, W. V.** (1982). *Methods of Logic* (4th edition). Harvard University Press.
- Quinlan, J. R.** (1979). Discovering rules from large collections of examples: A case study. In Michie, D. (a cura di), *Expert Systems in the Microelectronic Age*. Edinburgh University Press.
- Quinlan, J. R.** (1986). Induction of decision trees. *Machine Learning*, 1, 81–106.
- Quinlan, J. R.** (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
- Quinlan, S. e Khatib, O.** (1993). Elastic bands: Connecting path planning and control. In *ICRA-93*.
- Quirk, R., Greenbaum, S., Leech, G. e Svartvik, J.** (1985). *A Comprehensive Grammar of the English Language*. Longman.
- Rabani, Y., Rabinovich, Y. e Sinclair, A.** (1998). A computational view of population genetics. *Random Structures and Algorithms*, 12, 313–334.
- Rabiner, L. R. e Juang, B.-H.** (1993). *Fundamentals of Speech Recognition*. Prentice-Hall.
- Radford, A., Metz, L. e Chintala, S.** (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. e Sutskever, I.** (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. e Liu, P. J.** (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv:1910.10683.
- Rafferty, A. N., Brunskill, E., Griffiths, T. L. e Shafto, P.** (2016). Faster teaching via POMDP planning. *Cognitive Science*, 40, 1290–1332.
- Rahwan, T., Michalak, T. P., Wooldridge, M. e Jennings, N. R.** (2015). Coalition structure generation: A survey. *AIJ*, 229, 139–174.
- Raiert, M., Blankespoor, K., Nelson, G. e Playter, R.** (2008). Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41, 10822–10825.
- Rajpurkar, P., Zhang, J., Lopyrev, K. e Liang, P.** (2016). Squad: 100,000+ questions for machine comprehension of text. In *EMNLP-16*.

- Ramsey**, F. P. (1931). Truth and probability. In Braithwaite, R. B. (a cura di), *The Foundations of Mathematics and Other Logical Essays*. Harcourt Brace Jovanovich.
- Ramsundar**, B. e Zadeh, R. B. (2018). *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly.
- Rao**, D. A. S. e Verweij, G. (2017). Sizing the prize: What's the real value of AI for your business and how can you capitalise? PwC.
- Raphael**, B. (1976). *The Thinking Computer: Mind Inside Matter*. W. H. Freeman.
- Raphson**, J. (1690). *Analysis aequationum universalis*. Apud Abelem Swalle, London.
- Raschka**, S. (2015). *Python Machine Learning*. Packt.
- Rashevsky**, N. (1936). Physico-mathematical aspects of excitation and conduction in nerves. In *Cold Springs Harbor Symposia on Quantitative Biology. IV: Excitation Phenomena*.
- Rashevsky**, N. (1938). *Mathematical Biophysics: Physico-Mathematical Foundations of Biology*. University of Chicago Press.
- Rasmussen**, C. E. e Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.
- Rassenti**, S., Smith, V. e Bulfin, R. (1982). A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13, 402–417.
- Ratliff**, N., Bagnell, J. A. e Zinkevich, M. (2006). Maximum margin planning. In *ICML-06*.
- Ratliff**, N., Zucker, M., Bagnell, J. A. e Srinivasa, (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *ICRA-09*.
- Ratnaparkhi**, A. (1996). A maximum entropy model for part-of-speech tagging. In *EMNLP-96*.
- Ratner**, D. e Warmuth, M. (1986). Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In *AAAI-86*.
- Rauch**, H. E., Tung, F. e Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3, 1445–1450.
- Rayward-Smith**, V., Osman, I., Reeves, C. e Smith, G. (a cura di). (1996). *Modern Heuristic Search Methods*. Wiley.
- Real**, E., Aggarwal, A., Huang, Y. e Le, Q. V. (2018). Regularized evolution for image classifier architecture search. arXiv:1802.01548.
- Rechenberg**, I. (1965). Cybernetic solution path of an experimental problem. Library translation, Royal Aircraft Establishment.
- Regin**, J. (1994). A filtering algorithm for constraints of difference in CSPs. In *AAAI-94*.
- Reid**, D. B. (1979). An algorithm for tracking multiple targets. *IEEE Trans. Automatic Control*, 24, 843–854.
- Reif**, J. (1979). Complexity of the mover's problem and generalizations. In *FOCS-79*.
- Reiter**, R. (1980). A logic for default reasoning. *AIJ*, 13, 81–132.
- Reiter**, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V. (a cura di), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press.
- Reiter**, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.
- Renner**, G. e Ekart, A. (2003). Genetic algorithms in computer aided design. *Computer Aided Design*, 35, 709–726.
- Rényi**, A. (1970). *Probability Theory*. Elsevier.
- Resnick**, P. e Varian, H. R. (1997). Recommender systems. *CACM*, 40, 56–58.
- Rezende**, D. J., Mohamed, S. e Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *ICML-14*.
- Riazanov**, A. e Voronkov, A. (2002). The design and implementation of VAMPIRE. *AI Communications*, 15, 91–110.
- Ribeiro**, M. T., Singh, S. e Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. In *KDD-16*.
- Richardson**, M. e Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62, 107–136.
- Richter**, S. e Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In *ICAPS-09*.
- Ridley**, M. (2004). *Evolution*. Oxford Reader.
- Riley**, J. e Samuelson, W. (1981). Optimal auctions. *American Economic Review*, 71, 381–392.
- Riley**, P. (2019). Three pitfalls to avoid in machine learning. *Nature*, 572, 27–29.
- Riloff**, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *AAAI-93*.
- Ringgaard**, M., Gupta, R. e Pereira, F. (2017). SLING: A framework for frame semantic parsing. arXiv:1710.07032.
- Rintanen**, J. (1999). Improvements to the evaluation of quantified Boolean formulae. In *IJCAI-99*.

- Rintanen, J.** (2007). Asymptotically optimal encodings of conformant planning in QBF. In *AAAI-07*.
- Rintanen, J.** (2012). Planning as satisfiability: Heuristics. *AIJ*, 193, 45–86.
- Rintanen, J.** (2016). Computational complexity in automated planning and scheduling. In *ICAPS-16*.
- Ripley, B. D.** (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.
- Rissanen, J.** (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, IT-30, 629–636.
- Rissanen, J.** (2007). *Information and Complexity in Statistical Modeling*. Springer.
- Rivest, R.** (1987). Learning decision lists. *Machine Learning*, 2, 229–246.
- Robbins, H.** (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58, 527–535.
- Robbins, H.** e **Monro, S.** (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22, 400–407.
- Roberts, L. G.** (1963). Machine perception of three-dimensional solids. Technical report, MIT Lincoln Laboratory.
- Robertson, N.** e **Seymour, P. D.** (1986). Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7, 309–322.
- Robertson, S. E.** e **Sparck Jones, K.** (1976). Relevance weighting of search terms. *J. American Society for Information Science*, 27, 129–146.
- Robins, J.** (1986). A new approach to causal inference in mortality studies with a sustained exposure period: Application to control of the healthy worker survivor effect. *Mathematical Modelling*, 7, 1393–1512.
- Robinson, A.** e **Voronkov, A.** (a cura di). (2001). *Handbook of Automated Reasoning*. Elsevier.
- Robinson, J. A.** (1965). A machine-oriented logic based on the resolution principle. *JACM*, 12, 23–41.
- Robinson, S.** (2002). Computer scientists find unexpected depths in airfare search problem. *SIAM News*, 35(6).
- Roche, E.** e **Schabes, Y.** (a cura di). (1997). *Finite-State Language Processing*. Bradford Books.
- Rock, I.** (1984). *Perception*. W. H. Freeman.
- Rokicki, T.**, **Kociemba, H.**, **Davidson, M.** e **Dethridge, J.** (2014). The diameter of the Rubik's Cube group is twenty. *SIAM Review*, 56, 645–670.
- Rolf, D.** (2006). Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 1, 111–122.
- Rolnick, D.**, **Donti, P. L.**, **Kaack, L. H.**, et al. (2019). Tackling climate change with machine learning. arXiv:1906.05433.
- Rolnick, D.** e **Tegmark, M.** (2018). The power of deeper networks for expressing natural functions. In *ICLR-18*.
- Romanovskii, I.** (1962). Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3, 678–681.
- Ros, G.**, **Sellart, L.**, **Materzynska, J.**, **Vazquez, D.** e **Lopez, A. M.** (2016). The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR-16*.
- Rosenblatt, F.** (1957). The perceptron: A perceiving and recognizing automaton. Report, Project PARA, Cornell Aeronautical Laboratory.
- Rosenblatt, F.** (1960). On the convergence of reinforcement procedures in simple perceptrons. Report, Cornell Aeronautical Laboratory.
- Rosenblatt, F.** (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan.
- Rosenblatt, M.** (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27, 832–837.
- Rosenblueth, A.**, **Wiener, N.** e **Bigelow, J.** (1943). Behavior, purpose, and teleology. *Philosophy of Science*, 10, 18–24.
- Rosenschein, J. S.** e **Zlotkin, G.** (1994). *Rules of Encounter*. MIT Press.
- Rosenschein, S. J.** (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing*, 3, 345–357.
- Ross, G.** (2012). Fisher and the millionaire: The statistician and the calculator. *Significance*, 9, 46–48.
- Ross, S.** (2015). *A First Course in Probability* (9th edition). Pearson.
- Ross, S.**, **Gordon, G.** e **Bagnell, D.** (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS-11*.
- Rossi, F.**, **van Beek, P.** e **Walsh, T.** (2006). *Handbook of Constraint Processing*. Elsevier.
- Roth, D.** (1996). On the hardness of approximate reasoning. *AIJ*, 82, 273–302.
- Roussel, P.** (1975). Prolog: Manual de référence et d'utilisation. Tech. rep., Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.
- Rowat, P. F.** (1979). *Representing the Spatial Experience and Solving Spatial Problems in a Simulated Robot Environment*. Ph.D. thesis, University of British Columbia.

- Roweis**, S. T. e Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural Computation*, 11, 305–345.
- Rowley**, H., Baluja, S. e Kanade, T. (1998). Neural network-based face detection. *PAMI*, 20, 23–38.
- Roy**, N., Gordon, G. e Thrun, S. (2005). Finding approximate POMDP solutions through belief compression. *JAIR*, 23, 1–40.
- Rubin**, D. (1974). Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66, 688–701.
- Rubin**, D. (1988). Using the SIR algorithm to simulate posterior distributions. In Bernardo, J. M., de Groot, M. H., Lindley, D. V. e Smith, A. F. M. (a cura di), *Bayesian Statistics 3*. Oxford University Press.
- Rubinstein**, A. (1982). Perfect equilibrium in a bargaining model. *Econometrica*, 50, 97–109.
- Rubinstein**, A. (2003). Economics and psychology? The case of hyperbolic discounting. *International Economic Review*, 44, 1207–1216.
- Ruder**, S. (2018). NLP's ImageNet moment has arrived. *The Gradient*, July 8.
- Ruder**, S., Peters, M. E., Swayamdipta, S. e Wolf, T. (2019). Transfer learning in natural language processing. In *COLING-19*.
- Rumelhart**, D. E., Hinton, G. E. e Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.
- Rumelhart**, D. E. e McClelland, J. L. (a cura di). (1986). *Parallel Distributed Processing*. MIT Press.
- Rummery**, G. A. e Niranjan, M. (1994). On-line Q -learning using connectionist systems. Tech. rep., Cambridge University Engineering Department.
- Ruspini**, E. H., Lowrance, J. D. e Strat, T. M. (1992). Understanding evidential reasoning. *IJAR*, 6, 401–424.
- Russakovsky**, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. e Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *IJCV*, 115, 211–252.
- Russell**, J. G. B. (1990). Is screening for abdominal aortic aneurysm worthwhile? *Clinical Radiology*, 41, 182–184.
- Russell**, S. J. (1985). The complete guide to MRS. Report, Computer Science Department, Stanford University.
- Russell**, S. J. (1992). Efficient memory-bounded search methods. In *ECAI-92*.
- Russell**, S. J. (1998). Learning agents for uncertain environments. In *COLT-98*.
- Russell**, S. J. (1999). Expressive probability models in science. In *Proc. Second International Conference on Discovery Science*.
- Russell**, S. J. (2019). *Human Compatible*. Penguin.
- Russell**, S. J., Binder, J., Koller, D. e Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *IJCAI-95*.
- Russell**, S. J. e Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd edition). Prentice-Hall.
- Russell**, S. J. e Subramanian, D. (1995). Provably bounded-optimal agents. *JAIR*, 3, 575–609.
- Russell**, S. J. e Wefald, E. H. (1989). On optimal game-tree search using rational meta-reasoning. In *IJCAI-89*.
- Russell**, S. J. e Wefald, E. H. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press.
- Russell**, S. J. e Wolfe, J. (2005). Efficient belief-state AND-OR search, with applications to Kriegspiel. In *IJCAI-05*.
- Russell**, S. J. e Zimdars, A. (2003). Q-decomposition of reinforcement learning agents. In *ICML-03*.
- Rustagi**, J. S. (1976). *Variational Methods in Statistics*. Academic Press.
- Saad**, F. e Mansinghka, V. K. (2017). A probabilistic programming approach to probabilistic data analysis. In *NeurIPS 29*.
- Sabin**, D. e Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *ECAI-94*.
- Sabri**, K. E. (2015). Automated verification of role-based access control policies constraints using Prover9. arXiv:1503.07645.
- Sacerdoti**, E. D. (1974). Planning in a hierarchy of abstraction spaces. *AIJ*, 5, 115–135.
- Sacerdoti**, E. D. (1975). The nonlinear nature of plans. In *IJCAI-75*.
- Sacerdoti**, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier.
- Sadeghi**, F. e Levine, S. (2016). CAD2RL: Real single-image flight without a single real image. arXiv:1611.04201.
- Sadigh**, D., Sastry, S., Seshia, S. A. e Dragan, A. D. (2016). Planning for autonomous cars that leverage effects on human actions. In *Proc. Robotics: Science and Systems*.
- Sadler**, M. e Regan, N. (2019). *Game Changer*. New in Chess.
- Sadri**, F. e Kowalski, R. (1995). Variants of the event calculus. In *ICLP-95*.
- Sagae**, K. e Lavie, A. (2006). A best-first probabilistic shift-reduce parser. In *COLING-06*.
- Sahami**, M., Hearst, M. A. e Saund, E. (1996). Applying the multiple cause mixture model to text categorization. In *ICML-96*.

- Sahin**, N. T., Pinker, S., Cash, S. S., Schomer, D. e Halgren, E. (2009). Sequential processing of lexical, grammatical, and phonological information within Broca's area. *Science*, 326, 445–449.
- Sakuta**, M. e Iida, H. (2002). AND/OR-tree search for solving problems with uncertainty: A case study using screen-shogi problems. *Trans. Inf. Proc. Society of Japan*, 43, 1–10.
- Salomaa**, A. (1969). Probabilistic and weighted grammars. *Information and Control*, 15, 529–544.
- Samadi**, M., Felner, A. e Schaeffer, J. (2008). Learning from multiple heuristics. In *AAAI-08*.
- Samet**, H. (2006). *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.
- Sammut**, C., Hurst, S., Kedzier, D. e Michie, D. (1992). Learning to fly. In *ICML-92*.
- Samuel**, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3, 210–229.
- Samuel**, A. (1967). Some studies in machine learning using the game of checkers II—Recent progress. *IBM Journal of Research and Development*, 11, 601–617.
- Sanchez-Lengeling**, B., Wei, J. N., Lee, B. K., Gerkin, R. C., Aspuru-Guzik, A. e Wiltschko, A. B. (2019). Machine learning for scent: Learning generalizable perceptual representations of small molecules. arXiv:1910.10685.
- Sandholm**, T. (1999). Distributed rational decision making. In Weiß, G. (a cura di), *Multiagent Systems*. MIT Press.
- Sandholm**, T., Larson, K., Andersson, M., Shehory, O. e Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *AIJ*, 111, 209–238.
- Sandholm**, T. (1993). An implementation of the contract net protocol based on marginal cost calculations. In *AAAI-93*.
- Sang**, T., Beame, P. e Kautz, H. (2005). Performing Bayesian inference by weighted model counting. In *AAAI-05*.
- Sapir**, E. (1921). *Language: An Introduction to the Study of Speech*. Harcourt Brace Jovanovich.
- Sarawagi**, S. (2007). Information extraction. *Foundations and Trends in Databases*, 1, 261–377.
- Sargent**, T. J. (1978). Estimation of dynamic labor demand schedules under rational expectations. *J. Political Economy*, 86, 1009–1044.
- Sartre**, J.-P. (1960). *Critique de la Raison dialectique*. Editions Gallimard.
- Satia**, J. K. e Lave, R. E. (1973). Markovian decision processes with probabilistic observation of states. *Management Science*, 20, 1–13.
- Sato**, T. e Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *IJCAI-97*.
- Saul**, L. K., Jaakkola, T. e Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *JAIR*, 4, 61–76.
- Saunders**, W., Sastry, G., Stuhlmüller, A. e Evans, O. (2018). Trial without error: Towards safe reinforcement learning via human intervention. In *AAMAS-18*.
- Savage**, L. J. (1954). *The Foundations of Statistics*. Wiley.
- Savva**, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D. e Batra, D. (2019). Habitat: A platform for embodied AI research. arXiv:1904.01201.
- Sayre**, K. (1993). Three more flaws in the computational model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.
- Schaeffer**, J. (2008). *One Jump Ahead: Computer Perfection at Checkers*. Springer-Verlag.
- Schaeffer**, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P. e Sutphen, S. (2007). Checkers is solved. *Science*, 317, 1518–1522.
- Schank**, R. C. e Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum.
- Schank**, R. C. e Riesbeck, C. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum.
- Schapire**, R. E. e Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39, 135–168.
- Schapire**, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5, 197–227.
- Schapire**, R. E. (2003). The boosting approach to machine learning: An overview. In Denison, D. D., Hansen, M. H., Holmes, C., Mallick, B. e Yu, B. (a cura di), *Nonlinear Estimation and Classification*. Springer.
- Scharre**, P. (2018). *Army of None*. W. W. Norton.
- Schmid**, C. e Mohr, R. (1996). Combining grey-value invariants with local constraints for object recognition. In *CVPR-96*.
- Schmidhuber**, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85–117.
- Schofield**, M. e Thielscher, M. (2015). Lifting model sampling for general game playing to incomplete-information models. In *AAAI-15*.
- Schölkopf**, B. e Smola, A. J. (2002). *Learning with Kernels*. MIT Press.
- Schöning**, T. (1999). A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *FOCS-99*.

- Schoppers**, M. J. (1989). In defense of reaction plans as caches. *AIMag*, 10, 51–60.
- Schraudolph**, N. N., Dayan, P. e Sejnowski, T. (1994). Temporal difference learning of position evaluation in the game of Go. In *NeurIPS* 6.
- Schriftwieser**, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. e Silver, D. (2019). Mastering Atari, Go, chess and shogi by planning with a learned model. arXiv:1911.08265.
- Schröder**, E. (1877). *Der Operationskreis des Logikkalküls*. B. G. Teubner, Leipzig.
- Schulman**, J., Ho, J., Lee, A. X., Awwal, I., Bradlow, H. e Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Proc. Robotics: Science and Systems*.
- Schulman**, J., Levine, S., Abbeel, P., Jordan, M. I. e Moritz, P. (2015a). Trust region policy optimization. In *ICML-15*.
- Schulman**, J., Levine, S., Moritz, P., Jordan, M. e Abbeel, P. (2015b). Trust region policy optimization. In *ICML-15*.
- Schultz**, W., Dayan, P. e Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275, 1593.
- Schulz**, D., Burgard, W., Fox, D. e Cremers, A. B. (2003). People tracking with mobile robots using sample-based joint probabilistic data association filters. *Int. J. Robotics Research*, 22, 99–116.
- Schulz**, S. (2004). System Description: E 0.81. In *Proc. International Joint Conference on Automated Reasoning*, Vol. 3097 of *LNAI*.
- Schulz**, S. (2013). System description: E 1.8. In *Proc. Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*.
- Schütze**, H. (1995). *Ambiguity in Language Learning: Computational and Cognitive Models*. Ph.D. thesis, Stanford University. Also published by CSLI Press, 1997.
- Schwartz**, J. T., Scharir, M. e Hopcroft, J. (1987). *Planning, Geometry and Complexity of Robot Motion*. Ablex.
- Schwartz**, S. P. (a cura di). (1977). *Naming, Necessity, and Natural Kinds*. Cornell University Press.
- Scott**, D. e Krauss, P. (1966). Assigning probabilities to logical formulas. In Hintikka, J. e Suppes, P. (a cura di), *Aspects of Inductive Logic*. North-Holland.
- Searle**, J. R. (1980). Minds, brains, and programs. *BBS*, 3, 417–457.
- Searle**, J. R. (1990). Is the brain's mind a computer program? *Scientific American*, 262, 26–31.
- Searle**, J. R. (1992). *The Rediscovery of the Mind*. MIT Press.
- Sedgewick**, R. e Wayne, K. (2011). *Algorithms*. Addison-Wesley.
- Sefidgar**, Y. S., Agarwal, P. e Cakmak, M. (2017). Situated tangible robot programming. In *HRI-17*.
- Segaran**, T. (2007). *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly.
- Seipp**, J. e Röger, G. (2018). Fast downward stone soup 2018. IPC 2018 Classical Track.
- Seipp**, J., Sievers, S., Helmert, M. e Hutter, F. (2015). Automatic configuration of sequential planning portfolios. In *AAAI-15*.
- Selman**, B., Kautz, H. e Cohen, B. (1996). Local search strategies for satisfiability testing. In Johnson, D. S. e Trick, M. A. (a cura di), *Cliques, Coloring, and Satisfiability*. American Mathematical Society.
- Selman**, B. e Levesque, H. J. (1993). The complexity of path-based defeasible inheritance. *AIJ*, 62, 303–339.
- Selman**, B., Levesque, H. J. e Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *AAAI-92*.
- Seni**, G. e Elder, J. F. (2010). Ensemble methods in data mining: Improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2, 1–126.
- Seo**, M., Kembhavi, A., Farhadi, A. e Hajishirzi, H. (2017). Bidirectional attention flow for machine comprehension. In *ICLR-17*.
- Shachter**, R. D. (1986). Evaluating influence diagrams. *Operations Research*, 34, 871–882.
- Shachter**, R. D. (1998). Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *UAI-98*.
- Shachter**, R. D., D'Ambrosio, B. e Del Favero, B. A. (1990). Symbolic probabilistic inference in belief networks. In *AAAI-90*.
- Shachter**, R. D. e Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, 35, 527–550.
- Shachter**, R. D. e Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. In *UAI-98*.
- Shafer**, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.
- Shanahan**, M. (1997). *Solving the Frame Problem*. MIT Press.
- Shanahan**, M. (1999). The event calculus explained. In Wooldridge, M. J. e Veloso, M. (a cura di), *Artificial Intelligence Today*. Springer-Verlag.
- Shanahan**, M. (2015). *The Technological Singularity*. MIT Press.

- Shani**, G., Pineau, J. e Kaelbling, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27, 1–51.
- Shankar**, N. (1986). *Proof-Checking Metamathematics*. Ph.D. thesis, Computer Science Department, University of Texas at Austin.
- Shannon**, C. E. e Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press.
- Shannon**, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 41, 256–275.
- Shapley**, L. S. (1953a). A value for n -person games. In Kuhn, H. W. e Tucker, A. W. (a cura di), *Contributions to the Theory of Games*. Princeton University Press.
- Shapley**, S. (1953b). Stochastic games. *PNAS*, 39, 1095–1100.
- Sharai**, R. V. e Moir, T. J. (2016). An overview of applications and advancements in automatic sound recognition. *Neurocomputing*, 200, 22–34.
- Shatkay**, H. e Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. In *IJCAI-97*.
- Shazeer**, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E. e Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv:1701.06538.
- Shelley**, M. (1818). *Frankenstein: Or, the Modern Prometheus*. Pickering and Chatto.
- Sheppard**, B. (2002). World-championship-caliber scrabble. *AIJ*, 134, 241–275.
- Shi**, J. e Malik, J. (2000). Normalized cuts and image segmentation. *PAMI*, 22, 888–905.
- Shieber**, S. (1994). Lessons from a restricted Turing test. *CACM*, 37, 70–78.
- Shieber**, S. (a cura di). (2004). *The Turing Test*. MIT Press.
- Shimony**, S. E. (1994). Finding MAPs for belief networks is NP-hard. *AIJ*, 68, 399–410.
- Shoham**, Y. (1993). Agent-oriented programming. *AIJ*, 60, 51–92.
- Shoham**, Y. (1994). *Artificial Intelligence Techniques in Prolog*. Morgan Kaufmann.
- Shoham**, Y. e Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge Univ. Press.
- Shoham**, Y., Powers, R. e Grenager, T. (2004). If multi-agent learning is the answer, what is the question? In *Proc. AAAI Fall Symposium on Artificial Multi-Agent Learning*.
- Shortliffe**, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. Elsevier.
- Siciliano**, B. e Khatib, O. (a cura di). (2016). *Springer Handbook of Robotics* (2nd edition). Springer-Verlag.
- Sigaud**, O. e Buffet, O. (2010). *Markov Decision Processes in Artificial Intelligence*. Wiley.
- Sigmund**, K. (2017). *Exact Thinking in Demented Times*. Basic Books.
- Silberstein**, M., Weissbrod, O., Otten, L., Tzemach, A., Anisenia, A., Shtark, O., Tuberg, D., Galfrin, E., Gannon, I., Shalata, A., Borochowitz, Z. U., Dechter, R., Thompson, E. e Geiger, D. (2013). A system for exact and approximate genetic linkage analysis of SNP data in large pedigrees. *Bioinformatics*, 29, 197–205.
- Silva**, R., Melo, F. S. e Veloso, M. (2015). Towards table tennis with a quadrotor autonomous learning robot and onboard vision. In *IROS-15*.
- Silver**, D. e Veness, J. (2011). Monte-Carlo planning in large POMDPs. In *NeurIPS 23*.
- Silver**, D., Huang, A., Maddison, C. J., Guez, A. e Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 484–489.
- Silver**, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362, 1140–1144.
- Silver**, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. e Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550, 354–359.
- Silverman**, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.
- Silverstein**, C., Henzinger, M., Marais, H. e Moricz, M. (1998). Analysis of a very large AltaVista query log. Tech. rep., Digital Systems Research Center.
- Simmons**, R. e Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *IJCAI-95*.
- Simon**, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley.
- Simon**, H. A. (1947). *Administrative Behavior*. Macmillan.
- Simon**, H. A. (1963). Experiments with a heuristic compiler. *JACM*, 10, 493–506.
- Simon**, H. A. e Newell, A. (1958). Heuristic problem solving: The next advance in operations research. *Operations Research*, 6, 1–10.
- Simon**, J. C. e Dubois, O. (1989). Number of solutions to satisfiability instances—Applications to knowledge bases. *AIJ*, 3, 53–65.

- Simonis**, H. (2005). Sudoku as a constraint problem. In *CP-05 Workshop on Modeling and Reformulating Constraint Satisfaction Problems*.
- Singer**, P. W. (2009). *Wired for War*. Penguin Press.
- Singh**, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T. e Zhu, W. L. (2002). Open mind common sense: Knowledge acquisition from the general public. In *Proc. First International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*.
- Sisbot**, E. A., Marin-Urias, L. F., Alami, R. e Simeon, T. (2007). A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23, 874–883.
- Siskind**, J. M. e Pearlmuter, B. A. (2016). Efficient implementation of a higher-order language with built-in AD. arXiv:1611.03416.
- Sistla**, A. P. e Godefroid, P. (2004). Symmetry and reduced symmetry in model checking. *ACM Trans. Program. Lang. Syst.*, 26, 702–734.
- Sittler**, R. W. (1964). An optimal data association problem in surveillance theory. *IEEE Transactions on Military Electronics*, 8, 125–139.
- Skolem**, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, 4, 1–36.
- Skolem**, T. (1928). Über die mathematische Logik. *Norsk matematisk tidsskrift*, 10, 125–142.
- Slagle**, J. R. (1963). A heuristic program that solves symbolic integration problems in freshman calculus. *JACM*, 10.
- Slate**, D. J. e Atkin, L. R. (1977). CHESS 4.5— Northwestern University chess program. In Frey, P. W. (a cura di), *Chess Skill in Man and Machine*. Springer-Verlag.
- Slater**, E. (1950). Statistics for the chess computer and the factor of mobility. In *Symposium on Information Theory*. Ministry of Supply.
- Slocum**, J. e Sonneveld, D. (2006). *The 15 Puzzle*. Slocum Puzzle Foundation.
- Smallwood**, R. D. e Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21, 1071–1088.
- Smith**, B. (2004). Ontology. In Floridi, L. (a cura di), *The Blackwell Guide to the Philosophy of Computing and Information*. Wiley-Blackwell.
- Smith**, B., Ashburner, M., Rosse, C., et al. (2007). The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25, 1251–1255.
- Smith**, D. E., Genesereth, M. R. e Ginsberg, M. L. (1986). Controlling recursive inference. *AIJ*, 30, 343–389.
- Smith**, D. A. e Eisner, J. (2008). Dependency parsing by belief propagation. In *EMNLP-08*.
- Smith**, D. E. e Weld, D. S. (1998). Conformant Graphplan. In *AAAI-98*.
- Smith**, J. Q. (1988). *Decision Analysis*. Chapman and Hall.
- Smith**, J. E. e Winkler, R. L. (2006). The optimizer’s curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52, 311–322.
- Smith**, J. M. (1982). *Evolution and the Theory of Games*. Cambridge University Press.
- Smith**, J. M. e Szathmáry, E. (1999). *The Origins of Life: From the Birth of Life to the Origin of Language*. Oxford University Press.
- Smith**, M. K., Welty, C. e McGuinness, D. (2004). OWL web ontology language guide. Tech. rep., W3C.
- Smith**, R. G. (1980). *A Framework for Distributed Problem Solving*. UMI Research Press.
- Smith**, R. C. e Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *Int. J. Robotics Research*, 5, 56–68.
- Smith**, S. J. J., Nau, D. S. e Throop, T. A. (1998). Success in spades: Using AI planning techniques to win the world championship of computer bridge. In *AAAI-98*.
- Smith**, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3, 59–66.
- Smolensky**, P. (1988). On the proper treatment of connectionism. *BBS*, 2, 1–74.
- Smolensky**, P. e Prince, A. (1993). Optimality theory: Constraint interaction in generative grammar. Tech. rep., Department of Computer Science, University of Colorado at Boulder.
- Smullyan**, R. M. (1995). *First-Order Logic*. Dover.
- Smyth**, P., Heckerman, D. e Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9, 227–269.
- Snoek**, J., Larochelle, H. e Adams, R. P. (2013). Practical Bayesian optimization of machine learning algorithms. In *NeurIPS 25*.
- Solomonoff**, R. J. (1964). A formal theory of inductive inference. *Information and Control*, 7, 1–22, 224–254.
- Solomonoff**, R. J. (2009). Algorithmic probability—theory and applications. In Emmert-Streib, F. e Dehmer, M. (a cura di), *Information Theory and Statistical Learning*. Springer.

- Sondik**, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University.
- Sosic**, R. e Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, 6, 661–668.
- Sowa**, J. (1999). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Blackwell.
- Spaan**, M. T. J. e Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, 24, 195–220.
- Sparrow**, R. (2004). The Turing triage test. *Ethics and Information Technology*, 6, 203–213.
- Spiegelhalter**, D. J., Dawid, A. P., Lauritzen, S. e Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, 8, 219–282.
- Spirites**, P., Glymour, C. e Scheines, R. (1993). *Causation, Prediction, and Search*. Springer-Verlag.
- Spitkovsky**, V. I., Alshawi, H. e Jurafsky, D. (2010a). From baby steps to leapfrog: How less is more in unsupervised dependency parsing. In *NAACL HLT*.
- Spitkovsky**, V. I., Jurafsky, D. e Alshawi, H. (2010b). Profiting from mark-up: Hyper-text annotations for guided parsing. In *ACL-10*.
- Srivastava**, M. e Bickford, M. (1990). Formal verification of a pipelined microprocessor. *IEEE Software*, 7, 52–64.
- Srivastava**, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. e Salakhutdinov, R. (2014a). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15, 1929–1958.
- Srivastava**, S., Russell, S. J. e Ruan, P. (2014b). First-order open-universe POMDPs. In *UAI-14*.
- Staab**, S. (2004). *Handbook on Ontologies*. Springer.
- Stallman**, R. M. e Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *AIJ*, 9, 135–196.
- Stanfill**, C. e Waltz, D. (1986). Toward memory-based reasoning. *CACM*, 29, 1213–1228.
- Stanislawska**, K., Krawiec, K. e Vihma, T. (2015). Genetic programming for estimation of heat flux between the atmosphere and sea ice in polar regions. In *GECCO-15*.
- Stefik**, M. (1995). *Introduction to Knowledge Systems*. Morgan Kaufmann.
- Steiner**, D. F., MacDonald, R., Liu, Y., Truszkowski, P., Hipp, J. D., Gammage, C., Thng, F., Peng, L. e Stumpe, M. C. (2018). Impact of deep learning assistance on the histopathologic review of lymph nodes for metastatic breast cancer. *Am. J. Surgical Pathology*, 42, 1636–1646.
- Steinruecken**, C., Smith, E., Janz, D., Lloyd, J. e Ghahramani, Z. (2019). The Automatic Statistician. In Hutter, F., Kotthoff, L. e Vanschoren, J. (a cura di), *Automated Machine Learning*. Springer.
- Stergiou**, K. e Walsh, T. (1999). The difference all-difference makes. In *IJCAI-99*.
- Stickel**, M. E. (1992). A Prolog technology theorem prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104, 109–128.
- Stiller**, L. (1992). KQNKR. *J. International Computer Chess Association*, 15, 16–18.
- Stiller**, L. (1996). Multilinear algebra and chess endgames. In Nowakowski, R. J. (a cura di), *Games of No Chance*, MSRI, 29, 1996. Mathematical Sciences Research Institute.
- Stockman**, G. (1979). A minimax algorithm better than alpha-beta? *AIJ*, 12, 179–196.
- Stoffel**, K., Taylor, M. e Hendler, J. (1997). Efficient management of very large ontologies. In *AAAI-97*.
- Stone**, M. (1974). Cross-validatory choice and assessment of statistical predictions. *J. Royal Statistical Society*, 36, 111–133.
- Stone**, P. (2000). *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press.
- Stone**, P. (2003). Multiagent competitions and research: Lessons from RoboCup and TAC. In Lima, P. U. e Rojas, P. (a cura di), *RoboCup-2002: Robot Soccer World Cup VI*. Springer Verlag.
- Stone**, P. (2016). What's hot at RoboCup. In *AAAI-16*.
- Stone**, P., Brooks, R. A., Brynjolfsson, E., Calo, R., Etzioni, O., Hager, G., Hirschberg, J., Kalyanakrishnan, S., Kamar, E., Kraus, S., et al. (2016). Artificial intelligence and life in 2030. Tech. rep., Stanford University One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel.
- Stone**, P., Kaminka, G. e Rosenschein, J. S. (2009). Leading a best-response teammate in an ad hoc team. In *AAMAS Workshop in Agent Mediated Electronic Commerce*.
- Stone**, P., Sutton, R. S. e Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13, 165–188.
- Storvik**, G. (2002). Particle filters for state-space models with the presence of unknown static parameters. *IEEE Transactions on Signal Processing*, 50, 281–289.
- Strachey**, C. (1952). Logical or non-mathematical programmes. In *Proc. 1952 ACM National Meeting*.
- Stratonovich**, R. L. (1959). Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. *Radiotekhnika*, 2, 892–901.

- Stratonovich**, R. L. (1965). On value of information. *Izvestiya of USSR Academy of Sciences, Technical Cybernetics*, 5, 3–12.
- Sturtevant**, N. R. e Bulitko, V. (2016). Scrubbing during learning in real-time heuristic search. *JAIR*, 57, 307–343.
- Subramanian**, D. e Wang, E. (1994). Constraint-based kinematic synthesis. In *Proc. International Conference on Qualitative Reasoning*.
- Suk**, H.-I., Sin, B.-K. e Lee, S.-W. (2010). Hand gesture recognition based on dynamic Bayesian network framework. *Pattern Recognition*, 43, 3059–3072.
- Sun**, Y., Wang, S., Li, Y., Feng, S., Tian, H., Wu, H. e Wang, H. (2019). ERNIE 2.0: A continual pre-training framework for language understanding. arXiv:1907.12412.
- Sussman**, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier.
- Sutcliffe**, G. (2016). The CADE ATP system competition – CASC. *AIMag*, 37, 99–101.
- Sutcliffe**, G. e Suttner, C. (1998). The TPTP Problem Library: CNF Release v1.2.1. *JAR*, 21, 177–203.
- Sutcliffe**, G., Schulz, S., Claessen, K. e Gelder, A. V. (2006). Using the TPTP language for writing derivations and finite interpretations. In *Proc. International Joint Conference on Automated Reasoning*.
- Sutherland**, I. (1963). Sketchpad: A man-machine graphical communication system. In *Proc. Spring Joint Computer Conference*.
- Sutskever**, I., Vinyals, O. e Le, Q. V. (2015). Sequence to sequence learning with neural networks. In *NeurIPS 27*.
- Sutton**, C. e McCallum, A. (2007). An introduction to conditional random fields for relational learning. In Getoor, L. e Taskar, B. (a cura di), *Introduction to Statistical Relational Learning*. MIT Press.
- Sutton**, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9–44.
- Sutton**, R. S., McAllester, D. A., Singh, S. e Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS 12*.
- Sutton**, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML-90*.
- Sutton**, R. S. e Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd edition). MIT Press.
- Swade**, D. (2000). *Difference Engine: Charles Babbage And The Quest To Build The First Computer*. Diane Publishing Co.
- Sweeney**, L. (2000). Simple demographics often identify people uniquely. *Health (San Francisco)*, 671, 1–34.
- Sweeney**, L. (2002a). Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10, 571–588.
- Sweeney**, L. (2002b). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10, 557–570.
- Swerling**, P. (1959). First order error propagation in a stagewise smoothing procedure for satellite observations. *J. Astronautical Sciences*, 6, 46–52.
- Swift**, T. e Warren, D. S. (1994). Analysis of SLG-WAM evaluation of definite programs. In *Logic Programming: Proc. 1994 International Symposium*.
- Szegedy**, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. e Fergus, R. (2013). Intriguing properties of neural networks. arXiv:1312.6199.
- Szeliski**, R. (2011). *Computer Vision: Algorithms and Applications*. Springer-Verlag.
- Szepesvari**, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4, 1–103.
- Tadepalli**, P., Givan, R. e Driessens, K. (2004). Relational reinforcement learning: An overview. In *ICML-04*.
- Tait**, P. G. (1880). Note on the theory of the “15 puzzle”. *Proc. Royal Society of Edinburgh*, 10, 664–665.
- Tamaki**, H. e Sato, T. (1986). OLD resolution with tabulation. In *ICLP-86*.
- Tan**, P., Steinbach, M., Karpatne, A. e Kumar, V. (2019). *Introduction to Data Mining* (2nd edition). Pearson.
- Tang**, E. (2018). A quantum-inspired classical algorithm for recommendation systems. arXiv:1807.04271.
- Tarski**, A. (1935). Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1, 261–405.
- Tarski**, A. (1941). *Introduction to Logic and to the Methodology of Deductive Sciences*. Dover.
- Tarski**, A. (1956). *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press.
- Tash**, J. K. e Russell, S. J. (1994). Control strategies for a stochastic planner. In *AAAI-94*.
- Tassa**, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. arXiv:1801.00690.
- Tate**, A. (1975a). Interacting goals and their use. In *IJCAI-75*.
- Tate**, A. (1975b). *Using Goal Structure to Direct Search in a Problem Solver*. Ph.D. thesis, University of Edinburgh.
- Tate**, A. (1977). Generating project networks. In *IJCAI-77*.

- Tate**, A. e Whiter, A. M. (1984). Planning with multiple resource constraints and an application to a naval planning problem. In *Proc. First Conference on AI Applications*.
- Tatman**, J. A. e Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20, 365–379.
- Tattersall**, C. (1911). *A Thousand End-Games: A Collection of Chess Positions That Can be Won or Drawn by the Best Play*. British Chess Magazine.
- Taylor**, A. D. e Zwicker, W. S. (1999). *Simple Games: Desirability Relations, Trading, Pseudoweightings*. Princeton University Press.
- Taylor**, G., Stensrud, B., Eitelman, S. e Dunham, C. (2007). Towards automating airspace management. In *Proc. Computational Intelligence for Security and Defense Applications (CISDA) Conference*.
- Taylor**, P. (2009). *Text-to-Speech Synthesis*. Cambridge University Press.
- Tedrake**, R., Zhang, T. W. e Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *IROS-04*.
- Tellex**, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A., Teller, S. e Roy, N. (2011). Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI-11*.
- Tenenbaum**, J. B., Griffiths, T. L. e Niyogi, S. (2007). Intuitive theories as grammars for causal inference. In Gopnik, A. e Schulz, L. (a cura di), *Causal Learning: Psychology, Philosophy, and Computation*. Oxford University Press.
- Tesauro**, G. (1990). Neurogammon: A neural-network backgammon program. In *IJCNN-90*.
- Tesauro**, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8, 257–277.
- Tesauro**, G. (1995). Temporal difference learning and TD-Gammon. *CACM*, 38, 58–68.
- Tesauro**, G. e Galperin, G. R. (1997). On-line policy improvement using Monte-Carlo search. In *NeurIPS 9*.
- Tetlock**, P. E. (2017). *Expert Political Judgment: How Good Is It? How Can We Know?* Princeton University Press.
- Teyssier**, M. e Koller, D. (2005). Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *UAI-05*.
- Thaler**, R. (1992). *The Winner's Curse: Paradoxes and Anomalies of Economic Life*. Princeton University Press.
- Thaler**, R. e Sunstein, C. (2009). *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Penguin.
- Thayer**, J. T., Dionne, A. e Ruml, W. (2011). Learning inadmissible heuristics during search. In *ICAPS-11*.
- Theocharous**, G., Murphy, K. e Kaelbling, L. P. (2004). Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *ICRA-04*.
- Thiele**, T. (1880). Om anvendelse af mindste kvadraters methode i nogle tilfælde, hvor en komplikation af visse slags uensartede tilfældige fejlkilder giver fejlene en ‘systematisk’ karakter. *Vidensk. Selsk. Skr. 5. Rk., naturvid. og mat. Afd.*, 12, 381–408.
- Thielscher**, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *AIJ*, 111, 277–299.
- Thomas**, P. S., da Silva, B. C., Barto, A. G. e Brunskill, E. (2017). On ensuring that intelligent machines are well-behaved. arXiv:1708.05448.
- Thomaz**, A., Hoffman, G., Cakmak, M., et al. (2016). Computational human-robot interaction. *Foundations and Trends in Robotics*, 4, 105–223.
- Thompson**, K. (1986). Retrograde analysis of certain endgames. *J. International Computer Chess Association*, 9, 131–139.
- Thompson**, K. (1996). 6-piece endgames. *J. International Computer Chess Association*, 19, 215–226.
- Thompson**, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25, 285–294.
- Thorndike**, E. (1911). *Animal Intelligence*. Macmillan.
- Thornton**, C., Hutter, F., Hoos, H. H. e Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *KDD-13*.
- Thrun**, S., Burgard, W. e Fox, D. (2005). *Probabilistic Robotics*. MIT Press.
- Thrun**, S., Fox, D. e Burgard, W. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31, 29–53.
- Thrun**, S. (2006). Stanley, the robot that won the DARPA Grand Challenge. *J. Field Robotics*, 23, 661–692.
- Thrun**, S. e Pratt, L. (2012). *Learning to Learn*. Springer.
- Thurstone**, L. L. (1927). A law of comparative judgment. *Psychological Review*, 34, 273–286.
- Tian**, J., Paz, A. e Pearl, J. (1998). Finding a minimal d -separator. Tech. rep., UCLA Department of Computer Science.
- Tikhonov**, A. N. (1963). Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Dokl.*, 5, 1035–1038.

- Tipping**, M. E. e **Bishop**, C. M. (1999). Probabilistic principal component analysis. *J. Royal Statistical Society, 61*, 611–622.
- Titterington**, D. M., **Smith**, A. F. M. e **Makov**, U. E. (1985). *Statistical Analysis of Finite Mixture Distributions*. Wiley.
- Toma**, P. (1977). SYSTRAN as a multilingual machine translation system. In *Proc. Third European Congress on Information Systems and Networks: Overcoming the Language Barrier*.
- Tomasi**, C. e **Kanade**, T. (1992). Shape and motion from image streams under orthography: A factorization method. *IJCV, 9*, 137–154.
- Topol**, E. (2019). *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again*. Basic Books.
- Torralba**, A., **Fergus**, R. e **Weiss**, Y. (2008). Small codes and large image databases for recognition. In *CVPR*.
- Torralba**, A., **Linares López**, C. e **Borrajo**, D. (2016). Abstraction heuristics for symbolic bidirectional search. In *IJCAI-16*.
- Tramèr**, F., **Zhang**, F., **Juels**, A., **Reiter**, M. K. e **Ristenpart**, T. (2016). Stealing machine learning models via prediction APIs. In *USENIX Security Symposium*.
- Tran**, D., **Hoffman**, M., **Saurous**, R. A., **Brevdo**, E., **Murphy**, K. e **Blei**, D. M. (2017). Deep probabilistic programming. In *ICLR-17*.
- Trappenberg**, T. (2010). *Fundamentals of Computational Neuroscience* (2nd edition). Oxford University Press.
- Tsang**, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press.
- Tshitoyan**, V., **Dagdelen**, J., **Weston**, L., **Dunn**, A., **Rong**, Z., **Kononova**, O., **Persson**, K. A., **Ceder**, G. e **Jain**, A. (2019). Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature, 571*, 95.
- Tsitsiklis**, J. N. e **Van Roy**, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*, 674–690.
- Tukey**, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley.
- Tumer**, K. e **Wolpert**, D. (2000). Collective intelligence and Braess' paradox. In *AAAI-00*.
- Turian**, J., **Ratinov**, L. e **Bengio**, Y. (2010). Word representations: a simple and general method for semisupervised learning. In *ACL-10*.
- Turing**, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Mathematical Society, 2nd series, 42*, 230–265.
- Turing**, A. (1948). Intelligent machinery. Tech. rep., National Physical Laboratory. reprinted in (Ince, 1992).
- Turing**, A. (1950). Computing machinery and intelligence. *Mind, 59*, 433–460.
- Turing**, A., **Strachey**, C., **Bates**, M. A. e **Bowden**, B. V. (1953). Digital computers applied to games. In Bowden, B. V. (a cura di), *Faster than Thought*. Pitman.
- Turing**, A. (1947). Lecture to the London Mathematical Society on 20 February 1947.
- Turing**, A. (1996). Intelligent machinery, a heretical theory. *Philosophia Mathematica, 4*, 256–260. Scritto originale c. 1951.
- Tversky**, A. e **Kahneman**, D. (1982). Causal schemata in judgements under uncertainty. In Kahneman, D., Slovic, P. e Tversky, A. (a cura di), *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press.
- Tygar**, J. D. (2011). Adversarial machine learning. *IEEE Internet Computing, 15*, 4–6.
- Ullman**, J. D. (1985). Implementation of logical query languages for databases. *ACM Transactions on Database Systems, 10*, 289–321.
- Ullman**, S. (1979). *The Interpretation of Visual Motion*. MIT Press.
- Urmson**, C. e **Whittaker**, W. (2008). Self-driving cars and the Urban Challenge. *IEEE Intelligent Systems, 23*, 66–68.
- Valiant**, L. (1984). A theory of the learnable. *CACM, 27*, 1134–1142.
- Vallati**, M., **Chrpa**, L. e **Kitchin**, D. E. (2015). Portfolio-based planning: State of the art, common practice and open challenges. *AI Commun., 28*(4), 717–733.
- van Beek**, P. (2006). Backtracking search algorithms. In Rossi, F., van Beek, P. e Walsh, T. (a cura di), *Handbook of Constraint Programming*. Elsevier.
- van Beek**, P. e **Chen**, X. (1999). CPlan: A constraint programming approach to planning. In *AAAI-99*.
- van Beek**, P. e **Manchak**, D. (1996). The design and experimental analysis of algorithms for temporal reasoning. *JAIR, 4*, 1–18.
- van Bentham**, J. e **ter Meulen**, A. (1997). *Handbook of Logic and Language*. MIT Press.
- van den Oord**, A., **Dieleman**, S. e **Schrauwen**, B. (2014). Deep content-based music recommendation. In *NeurIPS 26*.

- van den Oord**, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. e Kavukcuoglu, K. (2016a). WaveNet: A generative model for raw audio. arXiv:1609.03499.
- van den Oord**, A., Kalchbrenner, N. e Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. arXiv:1601.06759.
- van Harmelen**, F., Lifschitz, V. e Porter, B. (2007). *The Handbook of Knowledge Representation*. Elsevier.
- van Heijenoort**, J. (a cura di). (1967). *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press.
- Van Hentenryck**, P., Saraswat, V. e Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(FD). *J. Logic Programming*, 37, 139–164.
- van Hoeve**, W.-J. (2001). The alldifferent constraint: a survey. In *6th Annual Workshop of the ERCIM Working Group on Constraints*.
- van Hoeve**, W.-J. e Katriel, I. (2006). Global constraints. In Rossi, F., van Beek, P. e Walsh, T. (a cura di), *Handbook of Constraint Processing*. Elsevier.
- van Lambalgen**, M. e Hamm, F. (2005). *The Proper Treatment of Events*. Wiley-Blackwell.
- van Nunen**, J. A. E. E. (1976). A set of successive approximation methods for discounted Markovian decision problems. *Zeitschrift fur Operations Research, Serie A*, 20, 203–208.
- Van Roy**, P. L. (1990). Can logic programming execute as fast as imperative programming? Report, Computer Science Division, UC Berkeley.
- Vapnik**, V. N. (1998). *Statistical Learning Theory*. Wiley.
- Vapnik**, V. N. e Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16, 264–280.
- Vardi**, M. Y. (1996). An automata-theoretic approach to linear temporal logic. In Moller, F. e Birtwistle, G. (a cura di), *Logics for Concurrency*. Springer.
- Varian**, H. R. (1995). Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*.
- Vasilache**, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S. e LeCun, Y. (2014). Fast convolutional nets with fbfft: A GPU performance evaluation. arXiv:1412.7580.
- Vaswani**, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. e Polosukhin, I. (2018). Attention is all you need. In *NeurIPS 30*.
- Veach**, E. e Guibas, L. J. (1995). Optimally combining sampling techniques for Monte Carlo rendering. In *Proc. 22rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*.
- Venkatesh**, S. (2012). *The Theory of Probability: Explorations and Applications*. Cambridge University Press.
- Vere**, S. A. (1983). Planning in time: Windows and durations for activities and goals. *PAMI*, 5, 246–267.
- Verma**, S. e Rubin, J. (2018). Fairness definitions explained. In *2018 IEEE/ACM International Workshop on Software Fairness*.
- Verma**, V., Gordon, G., Simmons, R. e Thrun, S. (2004). Particle filters for rover fault diagnosis. *IEEE Robotics and Automation Magazine*, June.
- Vinge**, V. (1993). The coming technological singularity: How to survive in the post-human era. In *Proc. Vision-21: Interdisciplinary Science and Engineering in the Era of Cyberspace*. NASA.
- Vinyals**, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Hassabis, D., Apps, C. e Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575, 350–354.
- Vinyals**, O., Ewalds, T., Bartunov, S. e Georgiev, P. (2017a). StarCraft II: A new challenge for reinforcement learning. arXiv:1708.04782.
- Vinyals**, O., Toshev, A., Bengio, S. e Erhan, D. (2017b). Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge. *PAMI*, 39, 652–663.
- Viola**, P. e Jones, M. (2004). Robust real-time face detection. *IJCV*, 57, 137–154.
- Visser**, U., Ribeiro, F., Ohashi, T. e Dellaert, F. (a cura di). (2008). *RoboCup 2007: Robot Soccer World Cup XI*. Springer.
- Viterbi**, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13, 260–269.
- Vlassis**, N. (2008). *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Morgan & Claypool.
- von Mises**, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit*. J. Springer.
- von Neumann**, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, 295–320.
- von Neumann**, J. e Morgenstern, O. (1944). *Theory of Games and Economic Behavior* (first edition). Princeton University Press.

- von Winterfeldt**, D. e Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press.
- Vossen**, T., Ball, M., Lotem, A. e Nau, D. S. (2001). Applying integer programming to AI planning. *Knowledge Engineering Review*, 16, 85–100.
- Wainwright**, M. e Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1, 1–305.
- Walker**, G. (1931). On periodicity in series of related terms. *Proc. Roy. Soc., A*, 131, 518–532.
- Walker**, R. J. (1960). An enumerative technique for a class of combinatorial problems. In *Proc. Sympos. Appl. Math.*, Vol. 10.
- Wallace**, A. R. (1858). On the tendency of varieties to depart indefinitely from the original type. *Proc. Linnean Society of London*, 3, 53–62.
- Walpole**, R. E., Myers, R. H., Myers, S. L. e Ye, K. E. (2016). *Probability and Statistics for Engineers and Scientists* (9th edition). Pearson.
- Walsh**, T. (2015). Turing’s red flag. arXiv:1510.09033.
- Waltz**, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H. (a cura di), *The Psychology of Computer Vision*. McGraw-Hill.
- Wang**, A., Prusachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O. e Bowman, S. R. (2019). SuperGLUE: A stickier benchmark for general-purpose language understanding systems. arXiv:1905.00537.
- Wang**, A., Singh, A., Michael, J., Hill, F., Levy, O. e Bowman, S. (2018a). GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv:1804.07461.
- Wang**, J., Zhu, T., Li, H., Hsueh, C.-H. e Wu, I.C. (2018b). Belief-state Monte Carlo tree search for phantom Go. *IEEE Transactions on Games*, 10, 139–154.
- Wanner**, E. (1974). *On Remembering, Forgetting and Understanding Sentences*. Mouton.
- Warren**, D. H. D. (1974). WARPLAN: A System for Generating Plans. Department of Computational Logic Memo, University of Edinburgh.
- Warren**, D. H. D. (1983). An abstract Prolog instruction set. Technical note, SRI International.
- Wasserman**, L. (2004). *All of Statistics*. Springer.
- Watkins**, C. J. (1989). *Models of Delayed Reinforcement Learning*. Ph.D. thesis, Psychology Department, Cambridge University.
- Watson**, J. D. e Crick, F. (1953). A structure for deoxyribose nucleic acid. *Nature*, 171, 737.
- Wattenberg**, M., Viégas, F. e Johnson, I. (2016). How to use t-SNE effectively. *Distill*, 1.
- Waugh**, K., Schnizlein, D., Bowling, M. e Szafron, C. (2009). Abstraction pathologies in extensive games. In *AAMAS-09*.
- Weibull**, J. (1995). *Evolutionary Game Theory*. MIT Press.
- Weidenbach**, C. (2001). SPASS: Combining superposition, sorts and splitting. In Robinson, A. e Voronkov, A. (a cura di), *Handbook of Automated Reasoning*. MIT Press.
- Weiss**, G. (2000a). *Multiagent Systems*. MIT Press.
- Weiss**, Y. (2000b). Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12, 1–41.
- Weiss**, Y. e Freeman, W. (2001). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13, 2173–2200.
- Weizenbaum**, J. (1976). *Computer Power and Human Reason*. W. H. Freeman.
- Weld**, D. S. (1994). An introduction to least commitment planning. *AIMag*, 15, 27–61.
- Weld**, D. S. (1999). Recent advances in AI planning. *AIMag*, 20, 93–122.
- Weld**, D. S., Anderson, C. R. e Smith, D. E. (1998). Extending Graphplan to handle uncertainty and sensing actions. In *AAAI-98*.
- Weld**, D. S. e de Kleer, J. (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann.
- Weld**, D. S. e Etzioni, O. (1994). The first law of robotics: A call to arms. In *AAAI-94*.
- Wellman**, M. P. (1985). Reasoning about preference models. Technical report, Laboratory for Computer Science, MIT.
- Wellman**, M. P. (1988). *Formulation of Tradeoffs in Planning under Uncertainty*. Ph.D. thesis, MIT.
- Wellman**, M. P. (1990a). Fundamental concepts of qualitative probabilistic networks. *AIJ*, 44, 257–303.
- Wellman**, M. P. (1990b). The STRIPS assumption for planning under uncertainty. In *AAAI-90*.
- Wellman**, M. P., Breese, J. S. e Goldman, R. (1992). From knowledge bases to decision models. *Knowledge Engineering Review*, 7, 35–53.
- Wellman**, M. P. e Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *ICAPS-92*.

- Wellman**, M. P., Wurman, P., O'Malley, K., Bangera, R., Lin, S., Reeves, D. e Walsh, W. (2001). Designing the market game for a trading agent competition. *IEEE Internet Computing*, 5, 43–51.
- Werbos**, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University.
- Werbos**, P. (1990). Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78, 1550–1560.
- Werbos**, P. (1992). Approximate dynamic programming for real-time control and neural modeling. In White, D. A. e Sofge, D. A. (a cura di), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold.
- Werbos**, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22, 25–38.
- Wesley**, M. A. e Lozano-Perez, T. (1979). An algorithm for planning collision-free paths among polyhedral objects. *CACM*, 22, 560–570.
- West**, D. M. (2018). *The Future of Work: Robots, AI, and Automation*. Brookings Institution Press.
- West**, S. M., Whittaker, M. e Crawford, K. (2019). Discriminating systems: Gender, race and power in AI. Tech. rep., AI Now Institute.
- Wexler**, Y. e Meek, C. (2009). MAS: A multiplicative approximation scheme for probabilistic inference. In *NeurIPS 21*.
- Wheatstone**, C. (1838). On some remarkable, and hitherto unresolved, phenomena of binocular vision. *Phil. Trans. Roy. Soc.*, 2, 371–394.
- White**, C., Neiswanger, W. e Savani, Y. (2019). BANANAS: Bayesian optimization with neural architectures for neural architecture search. arXiv:1910.11858.
- Whitehead**, A. N. e Russell, B. (1910). *Principia Mathematica*. Cambridge University Press.
- Whittle**, P. (1979). Discussion of Dr Gittins' paper. *J. Royal Statistical Society*, 41, 165.
- Whorf**, B. (1956). *Language, Thought, and Reality*. MIT Press.
- Widrow**, B. (1962). Generalization and information storage in networks of ADALINE “neurons”. In Yovits, M. C., Jacobi, G. T. e Goldstein, G. D. (a cura di), *Self-Organizing Systems*. Spartan.
- Widrow**, B. e Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record*.
- Wiedijk**, F. (2003). Comparing mathematical provers. In *Proc. 2nd Int. Conf. on Mathematical Knowledge Management*.
- Wiegley**, J., Goldberg, K., Peshkin, M. e Brokowski, M. (1996). A complete algorithm for designing passive fences to orient parts. In *ICRA-96*.
- Wiener**, N. (1942). The extrapolation, interpolation, and smoothing of stationary time series. Tech. rep., Research Project DIC-6037, MIT.
- Wiener**, N. (1948). *Cybernetics*. Wiley.
- Wiener**, N. (1950). *The Human Use of Human Beings*. Houghton Mifflin.
- Wiener**, N. (1960). Some moral and technical consequences of automation. *Science*, 131, 1355–1358.
- Wiener**, N. (1964). *God & Golem, Inc: A Comment on Certain Points Where Cybernetics Impinges on Religion*. MIT Press.
- Wilensky**, R. (1978). *Understanding Goal-Based Stories*. Ph.D. thesis, Yale University.
- Wilkins**, D. E. (1988). *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann.
- Wilkins**, D. E. (1990). Can AI planners solve practical problems? *Computational Intelligence*, 6, 232–246.
- Wilks**, Y. (2010). *Close Engagements With Artificial Companions: Key Social, Psychological, Ethical and Design Issues*. John Benjamins.
- Wilks**, Y. (2019). *Artificial Intelligence: Modern Magic or Dangerous Future*. Icon.
- Williams**, A., Nangia, N. e Bowman, S. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL HLT*.
- Williams**, B., Ingham, M., Chung, S. e Elliott, P. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. *Proc. IEEE*, 91(212–237).
- Williams**, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 229–256.
- Williams**, R. J. e Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1, 270–280.
- Williams**, R. J. e Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep., College of Computer Science, Northeastern University.
- Wilson**, D. H. (2011). *Robopocalypse*. Doubleday.
- Wilson**, R. A. e Keil, F. C. (a cura di). (1999). *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press.
- Wilson**, R. (2004). *Four Colors Suffice*. Princeton University Press.

- Wilt**, C. M. e Ruml, W. (2014). Speedy versus greedy search. In *Seventh Annual Symposium on Combinatorial Search*.
- Wilt**, C. M. e Ruml, W. (2016). Effective heuristics for suboptimal best-first search. *JAIR*, 57, 273–306.
- Wingate**, D. e Seppi, K. D. (2005). Prioritization methods for accelerating MDP solvers. *JMLR*, 6, 851–881.
- Wingate**, D., Stuhlmüller, A. e Goodman, N. D. (2011). Lightweight implementations of probabilistic programming languages via transformational compilation. In *AISTATS-11*.
- Winograd**, S. e Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise*. MIT Press.
- Winograd**, T. (1972). Understanding natural language. *Cognitive Psychology*, 3, 1–191.
- Winston**, P. H. (1970). Learning structural descriptions from examples. Technical report, Department of Electrical Engineering and Computer Science, MIT.
- Winternute**, S., Xu, J. e Laird, J. (2007). SORTS: A human-level approach to real-time strategy AI. In *Proc. Third Artificial Intelligence and Interactive Digital Entertainment Conference*.
- Winternitz**, L. (2017). Autonomous navigation above the GNSS constellations and beyond: GPS navigation for the magnetospheric multiscale mission and SEXTANT pulsar navigation demonstration. Tech. rep., NASA Goddard Space Flight Center.
- Witten**, I. H. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34, 286–295.
- Witten**, I. H. e Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37, 1085–1094.
- Witten**, I. H. e Frank, E. (2016). *Data Mining: Practical Machine Learning Tools and Techniques* (4th edition). Morgan Kaufmann.
- Witten**, I. H., Moffat, A. e Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd edition). Morgan Kaufmann.
- Wittgenstein**, L. (1922). *Tractatus Logico-Philosophicus* (2nd edition). Routledge and Kegan Paul. Reprinted 1971, edited by D. F. Pears and B. F. McGuinness.
- Wittgenstein**, L. (1953). *Philosophical Investigations*. Macmillan.
- Wojciechowski**, W. S. e Wojcik, A. S. (1983). Automated design of multiple-valued logic circuits by automated theorem proving techniques. *IEEE Transactions on Computers*, C-32, 785–798.
- Wolfe**, J. e Russell, S. J. (2007). Exploiting belief state structure in graph search. In *ICAPS Workshop on Planning in Games*.
- Wolpert**, D. (2013). Ubiquity symposium: Evolutionary computation and the processes of life: what the no free lunch theorems really mean: how to improve search algorithms. *Ubiquity*, December, 1–15.
- Wolpert**, D. e Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1), 67–82.
- Wong**, C., Houlsby, N., Lu, Y. e Gesmundo, A. (2019). Transfer learning with neural AutoML. In *NeurIPS 31*.
- Woods**, W. A. (1973). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*.
- Woods**, W. A. (1975). What's in a link? Foundations for semantic networks. In Bobrow, D. G. e Collins, A. M. (a cura di), *Representation and Understanding: Studies in Cognitive Science*. Academic Press.
- Wooldridge**, M. (2009). *An Introduction to MultiAgent Systems* (2nd edition). Wiley.
- Wooldridge**, M. e Rao, A. (a cura di). (1999). *Foundations of Rational Agency*. Kluwer.
- Wos**, L., Carson, D. e Robinson, G. (1964). The unit preference strategy in theorem proving. In *Proc. Fall Joint Computer Conference*.
- Wos**, L., Carson, D. e Robinson, G. (1965). Efficiency and completeness of the set-of-support strategy in theorem proving. *JACM*, 12, 536–541.
- Wos**, L., Overbeek, R., Lusk, E. e Boyle, J. (1992). *Automated Reasoning: Introduction and Applications* (2nd edition). McGraw-Hill.
- Wos**, L. e Robinson, G. (1968). Paramodulation and set of support. In *Proc. IRIA Symposium on Automatic Demonstration*.
- Wos**, L., Robinson, G., Carson, D. e Shalla, L. (1967). The concept of demodulation in theorem proving. *JACM*, 14, 698–704.
- Wos**, L. e Winker, S. (1983). Open questions solved with the assistance of AURA. In Bledsoe, W. W. e Loveland, D. (a cura di), *Automated Theorem Proving: After 25 Years*. American Mathematical Society.
- Wos**, L. e Pieper, G. (2003). *Automated Reasoning and the Discovery of Missing and Elegant Proofs*. Rinton Press.
- Wray**, R. E. e Jones, R. M. (2005). An introduction to Soar as an agent architecture. In Sun, R. (a cura di), *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*. Cambridge University Press.
- Wright**, S. (1921). Correlation and causation. *J. Agricultural Research*, 20, 557–585.

- Wright, S.** (1931). Evolution in Mendelian populations. *Genetics*, 16, 97–159.
- Wright, S.** (1934). The method of path coefficients. *Annals of Mathematical Statistics*, 5, 161–215.
- Wu, F. e Weld, D. S.** (2008). Automatically refining the Wikipedia infobox ontology. In *17th World Wide Web Conference (WWW2008)*.
- Wu, Y., Li, L. e Russell, S. J.** (2016a). SWIFT: Compiled inference for probabilistic programming languages. In *IJCAI-16*.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.** (2016b). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144.
- Wu, Y. e He, K.** (2018). Group normalization. arXiv:1803.08494.
- Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X. e Stolcke, A.** (2017). The Microsoft 2017 conversational speech recognition system. arXiv:1708.06073.
- Yampolskiy, R. V.** (2018). *Artificial Intelligence Safety and Security*. Chapman and Hall/CRC.
- Yang, G., Lin, Y. e Bhattacharya, P.** (2010). A driver fatigue recognition model based on information fusion and dynamic Bayesian network. *Inf. Sci.*, 180, 1942–1954.
- Yang, X.-S.** (2009). Firefly algorithms for multimodal optimization. In *International Symposium on Stochastic Algorithms*.
- Yang, X.-S. e Deb, S.** (2014). Cuckoo search: Recent advances and applications. *Neural Computing and Applications*, 24, 169–174.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R. e Le, Q. V.** (2019). XLNet: Generalized autoregressive pretraining for language understanding. arXiv:1906.08237.
- Yarowsky, D.** (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *ACL-95*.
- Ye, Y.** (2011). The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36, 593–784.
- Yedidia, J., Freeman, W. e Weiss, Y.** (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51, 2282–2312.
- Yeo, H.-S., Minami, R., Rodriguez, K., Shaker, G. e Quigley, A.** (2018). Exploring tangible interactions with radar sensing. *Proc. ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2, 1–25.
- Ying, C., Kumar, S., Chen, D., Wang, T. e Cheng, Y.** (2018). Image classification at supercomputer scale. arXiv:1811.06992.
- Yip, K. M.-K.** (1991). *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press.
- Yngve, V.** (1955). A model and an hypothesis for language structure. In Locke, W. N. e Booth, A. D. (a cura di), *Machine Translation of Languages*. MIT Press.
- Yob, G.** (1975). Hunt the wumpus! *Creative Computing*, Sep/Oct.
- Yoshikawa, T.** (1990). *Foundations of Robotics: Analysis and Control*. MIT Press.
- You, Y., Pan, X., Wang, Z. e Lu, C.** (2017). Virtual to real reinforcement learning for autonomous driving. arXiv:1704.03952.
- Young, H. P.** (2004). *Strategic Learning and Its Limits*. Oxford University Press.
- Young, S., Gašić, M., Thompson, B. e Williams, J.** (2013). POMDP-based statistical spoken dialog systems: A review. *Proc. IEEE*, 101, 1160–1179.
- Younger, D. H.** (1967). Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10, 189–208.
- Yu, D. e Deng, L.** (2016). *Automatic Speech Recognition*. Springer-Verlag.
- Yu, H.-F., Lo, H.-Y., Hsieh, H.-P. e Lou, J.-K.** (2011). Feature engineering and classifier ensemble for KDD Cup 2010. In *Proc. KDD Cup 2010 Workshop*.
- Yu, K., Sciuto, C., Jaggi, M., Musat, C. e Salzmann, M.** (2019). Evaluating the search phase of neural architecture search. arXiv:1902.08142.
- Yudkowsky, E.** (2008). Artificial intelligence as a positive and negative factor in global risk. In Bostrom, N. e Cirkovic, M. (a cura di), *Global Catastrophic Risk*. Oxford University Press.
- Yule, G. U.** (1927). On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers. *Phil. Trans. Roy. Soc., A*, 226, 267–298.
- Zadeh, L. A.** (1965). Fuzzy sets. *Information and Control*, 8, 338–353.
- Zadeh, L. A.** (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1, 3–28.

- Zaritskii**, V. S., Svetnik, V. B. e Shimelevich, L. I. (1975). Monte-Carlo technique in problems of optimal information processing. *Automation and Remote Control*, 36, 2015–22.
- Zeckhauser**, R. e Shepard, D. (1976). Where now for saving lives? *Law and Contemporary Problems*, 40, 5–45.
- Zeeberg**, A. (2017). D.I.Y. artificial intelligence comes to a Japanese family farm. *New Yorker*, August 10.
- Zelle**, J. e Mooney, R. (1996). Learning to parse database queries using inductive logic programming. In *AAAI-96*.
- Zemel**, R., Wu, Y., Swersky, K., Pitassi, T. e Dwork, C. (2013). Learning fair representations. In *ICML-13*.
- Zemelman**, B. V., Lee, G. A., Ng, M. e Miesenböck, G. (2002). Selective photostimulation of genetically chARGed neurons. *Neuron*, 33, 15–22.
- Zermelo**, E. (1913). Über Eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proc. Fifth International Congress of Mathematicians*.
- Zermelo**, E. (1976). An application of set theory to the theory of chess-playing. *Firbush News*, 6, 37–42. English translation of (Zermelo 1913).
- Zettlemoyer**, L. e Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI-05*.
- Zhang**, C., Bengio, S., Hardt, M., Recht, B. e Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv:1611.03530.
- Zhang**, H. e Stickel, M. E. (1996). An efficient algorithm for unit-propagation. In *Proc. Fourth International Symposium on Artificial Intelligence and Mathematics*.
- Zhang**, L., Pavlovic, V., Cantor, C. R. e Kasif, S. (2003). Human-mouse gene identification by comparative evidence integration and evolutionary analysis. *Genome Research*, 13, 1190–1202.
- Zhang**, N. L. e Poole, D. (1994). A simple approach to Bayesian network computations. In *Proc. 10th Canadian Conference on Artificial Intelligence*.
- Zhang**, S., Yao, L. e Sun, A. (2017). Deep learning based recommender system: A survey and new perspectives. arXiv:1707.07435.
- Zhang**, X., Zhao, J. e LeCun, Y. (2016). Character-level convolutional networks for text classification. In *NeurIPS 28*.
- Zhang**, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y. e Courville, A. (2017). Towards end-to-end speech recognition with deep convolutional neural networks. arXiv:1701.02720.
- Zhao**, K. e Huang, L. (2015). Type-driven incremental semantic parsing with polymorphism. In *NAACL HLT*.
- Zhou**, K., Doyle, J. e Glover, K. (1995). *Robust and Optimal Control*. Pearson.
- Zhou**, R. e Hansen, E. (2002). Memory-bounded A* graph search. In *Proc. 15th International FLAIRS Conference*.
- Zhou**, R. e Hansen, E. (2006). Breadth-first heuristic search. *AIJ*, 170, 385–408.
- Zhu**, B., Jiao, J. e Tse, D. (2019). Deconstructing generative adversarial networks. arXiv:1901.09465.
- Zhu**, D. J. e Latombe, J.-C. (1991). New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7, 9–20.
- Zhu**, J.-Y., Park, T., Isola, P. e Efros, A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV-17*.
- Zhu**, M., Zhang, Y., Chen, W., Zhang, M. e Zhu, J. (2013). Fast and accurate shift-reduce constituent parsing. In *ACL-13*.
- Ziebart**, B. D., Maas, A. L., Dey, A. K. e Bagnell, J. A. (2008). Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proc. 10th Int. Conf. on Ubiquitous Computing*.
- Ziebart**, B. D., Ratliff, N., Gallagher, G., Mertz, C., Peterson, K., Bagnell, J. A., Hebert, M., Dey, A. K. e Srinivasa, S. (2009). Planning-based prediction for pedestrians. In *IROS-09*.
- Zimmermann**, H.-J. (a cura di). (1999). *Practical Applications of Fuzzy Technologies*. Kluwer.
- Zimmermann**, H.-J. (2001). *Fuzzy Set Theory—And Its Applications* (4th edition). Kluwer.
- Zinkevich**, M., Johanson, M., Bowling, M. e Piccione, C. (2008). Regret minimization in games with incomplete information. In *NeurIPS 20*.
- Zipf**, G. (1935). *The Psychobiology of Language*. Houghton Mifflin.
- Zipf**, G. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley.
- Zobrist**, A. L. (1970). *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. Ph.D. thesis, University of Wisconsin.
- Zollmann**, A., Venugopal, A., Och, F. J. e Ponte, J. (2008). A systematic comparison of phrase-based, hierarchical and syntax-augmented statistical MT. In *COLING-08*.
- Zoph**, B. e Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv:1611.01578.
- Zuse**, K. (1945). The Plankalkül. Report, Gesellschaft für Mathematik und Datenverarbeitung.
- Zweig**, G. e Russell, S. J. (1998). Speech recognition with dynamic Bayesian networks. In *AAAI-98*.

Indice analitico

A

A prova di strategia 645
Accoppiamento debole 613
Acquisizione della conoscenza 278
Adversary argument 141
Agenti 6
-- basati su modello 54
-- basati su obiettivi 56
-- basati sull'utilità 57-58
-- basati sulla conoscenza 216-217
-- basati sulla logica proposizionale 244-253
-- capaci di apprendere 58-60
-- e ambienti 39-41
-- ibridi 248
-- intelligenti 39-64
-- logici 215-256
-- per ambienti parzialmente osservabili 137
-- per ricerca online e ambienti sconosciuti 140-146
-- razionali 6, 42
-- reattivi
---- basati su modello 54
---- semplici 52
-- risolutori di problemi 68-71
-- software 46
-- struttura 50-62
Aggiornamento di Bellman 584
Aggregazione 385
Agire
-- in condizioni di incertezza 397-400
-- razionalmente: l'approccio degli agenti razionali 6-7
-- umanamente: test di Turing 4-5

AI index 30
Albero
-- di gioco 153
-- di join 446
-- di ricerca 153
-- di ricerca and-or 128
Algebra lineare 661-662
Algoritmi 11, 667-669
-- Davis-Putnam 240
-- di backtracking completo 240
-- di clustering 446
-- di concatenazione all'indietro 301
-- di concatenazione in avanti 295
-- di Dijkstra 82
-- di eliminazione delle variabili 441
-- di ricerca 75-80
---- locale 242
---- minimax 155
---- per i giochi, limitazioni 178-179
---- stato-credenza incrementale 135
-- di Viterbi 483
-- evolutivi 121
-- forward-backward 480
-- genetici 121
-- online
---- per MDP 589
---- per POMDP 604
-- per la pianificazione classica 358-362
-- per MDP 583-591
-- per risolvere POMDP 601-605
-- pseudocodice 668
-- Rete 299

- semplificati basati su matrici 484
- sistemici 80
- ungherese 528
- Alleanza 157
- Alternanza di ricerca e inferenza 199
- Ambienti 46
 - ad agente singolo 47
 - competitivi 47
 - completamente osservabili 47
 - continui 48
 - cooperativi 47
 - deterministici 47
 - dinamici 48
 - discreti 48
 - episodici 48
 - ignoti 49
 - inosservabili 47
 - multiagente 47
 - proprietà 609-615
 - natura 45-50
 - noti 49
 - operativi 45
 - parzialmente osservabili 47
 - proprietà 47
 - semidinamici 48
 - sequenziali 48
 - specificazione 45
 - statici 48
 - stocastici 48
- Amministratore 642
- Analisi
 - asintotica 659, 660
 - degli approcci alla pianificazione 388
 - di complessità 659-660
 - di sensibilità 563
 - e decisioni robuste 563
- Annuncio del compito 642
- Apprendimento 44, 317
 - automatico 4, 27-28
 - raccomandazioni 32
 - dei vincoli 202
 - di metalivello 109
 - hebbiano 20
 - nella ricerca online 146
- Approccio
 - dichiarativo 217
 - procedurale 217
- Approssimazione conservativa 250
- Architettura 50
 - cognitiva 300-306
- Arco di persistenza 499
- Arità 264
- Ascesa più ripida 116
- Assegnamento 186
 - completo 186
 - consistente 186
 - parziale 186
- Asserzioni 272
 - nella logica del primo ordine 272
- Assicurazione auto 436
- Assiomi 216
 - da effetto 246
 - della teoria della probabilità 405
 - di esclusione di azioni 252
 - di frame 246
 - di Kolmogorov 406
 - di Peano 274
 - di precondizione 252
 - di stato successore 247
- Associazione dei dati 526
- Assunzione 346
- Asta 643
 - a offerta segreta 645
 - al rialzo 644
 - al secondo prezzo a offerta segreta 645
 - di Vickrey 645
 - efficiente 644
 - inglese 644
- Astrazione 70
 - di stato 365
 - nella pianificazione 365
 - livello 70
- ATMS 345
- Atomo 266
- Attacco sybil 519
- Attitudine proposizionale 335
- Attore 611
- Attributi 61
- Attuatore 46
- Autonomia 45
- Avversione all'ambiguità 550

- Avversità al rischio 547
- Azioni 69
 - applicabili 69
 - congiunte 613
 - di alto livello 366
 - esecuzione 69
 - irreversibili 141
 - preferite 364
 - primitive 366
 - rilevanti 359
- B**
- Backjumping 200
 - guidato dai conflitti 201
- Backtracking
 - cronologico 200
 - intelligente 200
- Bandito
 - con n braccia 591
 - con un braccio solo 593
- Base di conoscenza 216, 227
- Base di Herbrand 312
- Basi matematiche 659-665
- Behaviorista, movimento 16
- Bellman, aggiornamento 584
- Benefici dimostrabili 8
- Benessere sociale 619
 - egalitario 620
 - utilitaristico 620
- Bernoulli 403
- Bicondizionale 225
- Big data 28-29
- Bluff 177
- BNF (forma di Backus-Naur) 667
- C**
- Caccia al tesoro 562
- Calcolo
 - degli eventi 332
 - dell'indice di Gittins 594
 - delle situazioni 361
- Cammino 70
 - ciclico 79
 - critico 385
 - ridondante 79
- Campionamento 665
 - a blocchi 458
 - di Gibbs 453
 - di importanza 450
 - sequenziale 502
 - di rigetto 448
 - di Thompson 596
- Caratteristica 162
- Caso condizionante 425
- Categorie 326
 - disgiunte 327
 - e oggetti 326-331
- Catena di Markov 453
- Centro 642
- Chiusura
 - del dominio 271
 - della risoluzione 235
- Cibernetica 18
- Ciclo 79
- Circoscrizione
 - con priorità 343
 - e logica di default 342
- Classe di ambienti 50
- Classificazione 341
 - di testi 415
 - con un modello Bayes ingenuo 415
- Clausola 233
 - definita 237
 - del primo ordine 294
 - di Horn 237
 - e clausole definite 236
 - obiettivo 237
 - unitaria 233
- Clustering 446
- Clutter 527
- CNF 234
- Co-NP 661
- Co-NP-completo 661
- Coalizione 636

- Coda 78
---con priorità 78
---FIFO 78
---LIFO 78
Coefficiente di Gini 620
Collegamento procedurale 340
Collusione 644
Colorazione di una mappa 186
Commutatività 197
Compattezza
---e ordinamento dei nodi 428
Compilare inferenza approssimata 460
Complessità
---relativa ai dati 298
---spaziale 80
---temporale 80
Completamento 305
Completezza 80, 223
---della risoluzione 311
---per refutazione 311
Componenti connessi 204
Composizionalità 258
Composizione fisica di oggetti 328
Computabilità 11
Computing quantistico 17
Concatenazione
---all'indietro 237-306
---in avanti 237, 294-301
----efficiente 297
Concessione 653
Concetto di soluzione 616
Conclusione 225
Concorrenza 612
---vera 612
Condizionamento 408
---con insieme di taglio 205, 207
Condorcet, paradosso 648
Configurazione VLSI 74, 77
Confini di ricerca 94
Confronto tra le strategie di ricerca non informata 88
Congettura della soglia di soddisficiabilità 244
Congiunzione 225
Connessioni multiple 444
Connessionisti, modelli 26
Connettivo logico 224
Conoscenza iniziale 216
Conseguenza logica 221
Consequenzialismo 42
Consistente 448
Consistenza 93, 341
---d'arco 191
---orientato 205
---di cammino 193
---di nodo 191
---locale 191
Constraint weighting 204
Consumabile 384
Conteggio
---con modello ponderato 445
---di Borda 649
Continuità 542
Contract net protocol 642
Contraddizione 230
Contrattazione 650
Contrazione 584
Contributo marginale 638
Controllo di occorrenza 291
Controparte 610
Convenzione 615
Coperta di Markov 430
Corpo 237
Correttezza 223
Cose 331
Costante di Skolem 288
Costo-opportunità 598
Costruzione di piani mediante inferenza proposizionale 250
Covarianza 665
Credenze e desideri in condizioni di incertezza 540-541
Cresta 118
Criptoaritmetica 189
Criterio back-door 464
CSP binario 189

D

- D-separazione 431
- Database
 - deduttivo 300
 - di pattern 106
 - disgiunti 107
- Datalog 295
- DBN 496
- Debolmente dominante 617
- Decisioni
 - collettive 642-654
 - complesse 573-608
 - minimax 155
 - multiagente 609-658
 - ottime
 - nei giochi 154-161
 - nei giochi multiplayer 156
 - robuste 564
 - semplici 539-571
 - strategiche 610
- Decisore singolo 609
- Decisori multipli 610
- Deep learning 29
- Delega agli esseri umani 565
- Demodulazione 315
- Deontologia 10
- Descrizione
 - ottimistica 371
 - pessimistica 372
- Deviazione standard 664
- Diagramma di influenza 556
- Diametro 86
- Dichiarazione
 - dei tipi 514
 - numerica 519
- Dilemma del prigioniero 617
- Dimensione del passo 126
- Dimostrazione 230
 - di teoremi 228-239
 - non costruttiva 311
 - per assurdo 230
 - per risoluzione 232
- Direzioni
 - causale 412
 - diagnostica 412
- Discendente 430
- Discretizzazione 125, 433
- Disgiunzione 225
- Distanza
 - in linea d'aria 90
 - Manhattan 103
- Distribuzione
 - a ordini di grandezza 520
 - canonica 431
 - categoriale 404
 - condizionata
 - rappresentazione efficiente 431
 - di Poisson 520
 - di probabilità 404, 663-665
 - congiunta 404
 - congiunta completa 405
 - gaussiana 663
 - aggiornamento 490
 - multivariata 664
 - log-normale discreta 520
 - normale standardizzata 664
 - proposta 458
 - adattiva 534
 - stazionaria 455
 - uniforme 665
- Disuguaglianza triangolare 93
- do-calculus 463
- Dominanza 552
 - stocastica 552
 - stretta 552
- Dominazione 104
- Dominio 262, 272
 - continuo 189
 - dei circuiti elettronici 280
 - della parentela 273
 - discreto 188
 - elementi 262
 - finito 188
 - infinito 189
 - orientato ai compiti 652
- Dualismo 9
- Durata 384

E

Economia 12-13, 152
Effetto 354
---ancoraggio 550
---certezza 549
---condizionale 378
---framing 550
---mancante 381
---orizzonte 164
Elemento
---critico 59
---di apprendimento 58
---esecutivo 58
Elicitazione delle preferenze 544
Eliminazione
---degli and 231
---delle variabili 441
Elitismo 121
Empirismo 9
Enunciati osservativi 9
Equazione
---di Bellman 579
---strutturale 461
Equilibrio
---con strategie dominanti 617
---dettagliato 455
---di Bayes-Nash 632
---di Nash 618
---maximin 623
---perfetto di Nash nei sottogiochi 629
Equivalente certo 547
Equivalenza logica 229
Ereditarietà 326
---multipla 339
Ergodico 455
Esecuzione interleaved 612
Esiti 399
---confittuali 651
Esperimento casuale controllato 464
Esplorazione 44, 167
Espressività 62
Estensione
---di una teoria 344
---singola 165
Esternalità 647
Estremi consistenti 195

Estrinseco 331
Euristica
---ammissibile 91
---apprendere dall'esperienza 109
---che ignora le precondizioni 362
---di grado 199
---differenziale 108
---e prestazioni 103
---generazione
----con punti di riferimento (landmark) 107
----da problemi rilassati 104
----da sottoproblemi: database di pattern 106
---inammissibile 95
---MRV 198
---per la pianificazione 362-366
Eventi 332-335, 401, 438
---calcolo 332
---esogeni 381
Evidenza 402
Evoluzione automatica 24
Expit 435

F

Fallimento transitorio 498
Falso allarme 527
Fare la cosa giusta 7
Fatto 237
Fattore 441
---di certezza 25
---di ramificazione 80
---effettivo 103
---di sconto 577
Fattorizzazione 233
Filosofia 8-10
Filtraggio 475
---e predizione 476
Filtro 138
---del vicino più prossimo 528
---di Kalman 489-496
----a commutazione 495
----applicabilità 494
----esteso (EKF) 495
Fluente 245, 334
---mancante 381

- Folk theorem di Nash 627
 Forma estesa 627
 Forma normale congiuntiva 234
 -- per la logica del primo ordine 307
 Forma sequenziale 631
 Formula 216
 -- atomica 224, 266
 -- complessa 224, 267
 -- generale per l'informazione perfetta 559
 Formulazione
 -- a stato completo 117
 -- dell'obiettivo 68
 -- di problemi 68-70
 Fortemente dominante 617
 Fortemente k-consistente 193
 Forzare il mondo in uno stato 132
 Frame 26
 Frontiera 76
 Funzione 260
 -- agente 40
 -- azione-utilità 557
 -- caratteristica 635
 -- di benessere sociale 648
 -- di costo 19
 -- dell'azione 70
 -- di densità di probabilità 404, 663
 -- di distribuzione cumulata 663
 -- di payoff 616
 -- di scelta sociale 648
 -- di Skolem 308
 -- di utilità 57, 540, 544-551
 -- moltiplicativa 555
 -- multiattributo 551-556
 -- ordinale 543
 -- di valutazione 77, 152, 162
 -- per giochi con elementi casuali 172
 -- euristica 88, 102-109
 -- lineare pesata 162
 -- logistica 435
 -- obiettivo 116
 -- origine 520
 -- totale 263
 -- unità 329
 -- valore 543
 -- additiva 555
 -- Q 579
- G**
 Gaussiana
 -- condizionata 435
 -- lineare 434
 Generatore 669
 -- di problemi 59
 Gerarchia tassonomica 326
 Gini, coefficiente 620
 Giocatori 615
 -- fintizi 639
 -- simmetrici 639
 Giochi 32
 -- a somma zero 152, 621
 -- a due giocatori 152
 -- base 624
 -- con una sola mossa 615
 -- cooperativi 611, 635-642
 -- computazioni 639
 -- decisioni ottime 154-161
 -- delle graffette 633
 -- dell'ultimatum 651
 -- di assistenza 36
 -- di carte 176
 -- in forma normale 615
 -- non cooperativi 611, 615-635
 -- parzialmente osservabili 173-178
 -- ripetuti 623
 -- sequenziali 627
 -- stocastici 170-173
 Giudizio umano, e irrazionalità 549
 Giustificazione 345
 Gorilla, problema 36
 Gradiente 125
 -- empirico 125
 Grado
 -- di credenza 399
 -- di verità 261
 Grafo 70
 -- delle strutture di coalizioni 641
 -- dello spazio degli stati 153
 -- di pianificazione 361
 -- di vincoli 187
 -- duale 190
 -- esistenziale 338
 -- morale 431
 Grande coalizione 636
 Grounding 223, 518

H

Hessiana 126
Hill climbing 116
---con prima scelta 119
---con riavvio casuale 119
---stocastico 119

I

IA a livello umano 35
Ignoranza
---pratica 398
---teorica 398
Imaging cerebrale 5
Impegno
---epistemologico 262
---ontologico 261
Implementazione 367
---di un agente che raccoglie informazioni 561
Implicazione 225
Imputazione 637
Incentivo 611
Incertezza
---dell'esistenza 519
---dell'identità 519
---e decisioni razionali 399
---e tempo 472-475
---quantificazione 397-422
---relazionale 516
---riassunto 398
---sulle proprie preferenze 564
Indice
---di deviazione 95
---di Gittins 594
Indicizzazione 292
---dei predicati 292
Indipendenza 410-411
---condizionale 413
---dei sotto-obiettivi 365
---delle preferenze 554
---mutua per la preferenza 555
---nelle reti bayesiane 430
---specifica del contesto 431
---tra utilità 555
Individualmente razionale 653
Individuazione 331
Induzione 9
---a ritroso 624
Inferenza 216
---approssimata
---nelle DBN 501
---nelle reti bayesiane 447-460
---basata su distribuzioni congiunte complete 407-409
---e dimostrazioni 230
---esatta
---complessità 444
---nelle DBN 500
---nelle reti bayesiane 438-446
---logica 223
---mediante simulazione con catene di Markov 453
---nei CSP 191-196
---nei modelli probabilistici a universo aperto 521
---nei modelli relazionali di probabilità 517
---nei modelli temporali 475-483
---nei programmi generativi 533
---nella logica del primo ordine 287-321
---per enumerazione 439
---probabilistica 407
---proposizionale, e inferenza del primo ordine 287
---ridondante, e cicli infiniti 303
---risultati 532
Information gathering 44
Informazione
---imperfetta 152, 629
---perfetta 152, 559, 627
---valore 558-564
Ingegneria
---della conoscenza 278
---nella logica del primo ordine 278-283
---processo 278
---informatica 17-18
---ontologica 324-326
Insieme
---convesso 127
---dei conflitti 200
---di negoziazione 651
---di supporto 316
---di taglio dei cicli 206
---informativo 629
---raggiungibile 370

Intelligenza artificiale

- cos'è 4-8
- fondamenti 8-19
- generale 35
- in climatologia 33
- in medicina 33
- rischi e opportunità 33-37
- stato dell'arte 30-33
- storia 19-29

Intelligenza aumentata 16

Interazione uomo-computer 16

Interfacce cervello-macchina 15

Interpretazione 265

- del linguaggio naturale 4
- delle immagini 33
- estesa 267
- intesa 265

Interrogazione 407

Intervallo 403

Intrinsico 331

Introspezione 5

Ipergrafo di vincoli 189

Ipotesi

- dei nomi unici 271
- del mondo chiuso 271
- di Markov 473
- sensoriale 474

Istanza 123

Istantiazione

- esistenziale 288
- universale 288

Iterazione

- dei valori 583
- delle politiche 587
- modificata 588

J

Job 384

Job-shop scheduling 384

JTMS 345

K

- K-consistenza 193
- Kriegspiel 174

L

- Larghezza d'albero 208
- Lemma di lifting 313
- Letterali 225
 - complementari 233
- Lettura di testo 530
- Lifting 290
- Limite
 - della media 625
 - superiore di confidenza 596
- Line search 126
- Linguaggi 667-669
 - del pensiero 258
 - delle proposizioni, nelle asserzioni probabilistiche 403
 - di programmazione probabilistici (PPL) 512
 - sintassi e semantica 513
 - di rappresentazione 257-262
 - della conoscenza 216
 - forma di Backus-Naur (BNF) 667
 - formali e naturali 260
 - Linguistica 19
 - computazionale 19
- Lisp 22
- Lista
 - di aggiunte 355
 - di eliminazioni 355
 - ignorare 363
 - di legami 272
- Livello
 - dell'implementazione 217
 - della conoscenza 217
- Località 247
- Localizzazione 138
- Localmente strutturato 428
- Locomozione su arti 31
- Logica 6, 221-224
 - del primo ordine 257-285
 - uso 272-278
 - descrittiva 338, 340

- di default 343
- di ordine superiore 262
- formale 11
- fuzzy 261
- modale 336
- temporale 262
 - lineare 337
- non monotona 343
- proposizionale 224-228
 - dimostrazione di teoremi 228-239
- Logit inverso 435
- Lotteria 541
- standard 544
- LRTA (*learning real-time A**) 145

- M**
- MA (*memory-bounded A**) 100
- MAC 200
- Macchine che portano benefici 7-8
- Magic set 300
- Makespan 384
- Maledizione dell'ottimizzatore 548
- Mancato rilevamento 527
- Marginalizzazione 408
- Margine 386
 - minimo 387
- Massima Utilità Attesa (MEU) 400
- Massimo
 - globale 116
 - locale 118
- Matching pennies 619
- Matematica 10-12
- Matrice 661-662
 - dei guadagni di Kalman 493
 - dei payoff 616
 - di covarianza 665
 - di osservazione 484
 - identità 662
 - inversa 662
 - singolare 662
 - trasposta 662
- Maximin 621
- MDP
 - algoritmi 583-591
 - di ripartenza 595
 - parzialmente osservabili 598-601
- Meganodo 446
- Memorizzazione e recupero di informazioni 292
- Metaragionamento 179
- Metaregola 306
- Metodi
 - deboli 24
 - del cammino critico 385
 - di campionamento diretto 447
 - per costruire reti bayesiane 427
- Metropolis-Hastings 453
- Micromondi 22
- Micromort 545
- Miglior risposta 618
 - miope 620
- Min-conflicts 203
- Minaccia credibile 629
- Minimax 154
- Minimizzazione logica 329
- Minimo globale 116
- Miope, acquisizione di informazione 561
- Misurare le prestazioni nella risoluzione di problemi 80
- Misure 329
 - di prestazione 42, 45
- Model checking 223
 - proposizionale efficiente 239-244
- Modellazione cognitiva 5
- Modelli 221
 - di Bayes ingenui 414-416
 - di contrattazione a offerte alternate 650
 - di errore gaussiano 498
 - di fallimento
 - persistente 499
 - transitorio 498
 - di Markov
 - incorporamento 533
 - nascosti 483-489
 - localizzazione 486
 - di probabilità 401
 - di transizione 54, 69, 153, 473

- nascosti di Markov 28
 - per la logica del primo ordine 262
 - probabilistici 663
 - a universo aperto 518-525
 - relazionali di probabilità 512-518
 - sensoriali 54, 473
 - standard 7
 - Modus Ponens 230
 - generalizzato 290
 - Mondo
 - dei blocchi 22, 356
 - del wumpus 218-221, 276
 - rivisitato 416-419
 - dell'aspirapolvere erratico 127
 - possibile 221, 336
 - Monitoraggio 138
 - del piano 382
 - del traffico 529
 - dell'azione 382
 - dell'esecuzione 381
 - dell'obiettivo 382
 - Monotonicità 94, 232, 342, 542
 - Monte Carlo 447
 - per catene di Markov 453
 - Moore, legge di 17
 - Mossa 152
 - killer 160
 - laterale 118
 - ordinamento 159
 - Mucchio 328
 - Multiattore 611
 - Multiplexer 516
 - Multitasking 597
 - Mutuamente indipendente per l'utilità 555
 - No-good 202
 - Nodi 75
 - and 129
 - certamente espansi 95
 - deterministici 431
 - di casualità 170, 556
 - di decisione 557
 - di utilità 557
 - espandere 75
 - figli 75
 - generare 75
 - or 129
 - padri 75
 - pozzo 432
 - raggiunti 76
 - successori 75
 - Non determinismo
 - angelico 370
 - demoniaco 370
 - Non monotonicità 342
 - Nonparametriche, rappresentazioni 433
 - Norma
 - del massimo 585
 - sociale 615
 - Notazione
 - base della teoria della probabilità 401-407
 - infissa 275
 - O() 659
 - prefissa 275
 - NP 660
 - e problemi intrinsecamente difficili 660
 - NP-completezza 12
 - NP-completo 660
 - NP-difficile 661
 - Nucleo 637
 - di transizione 455
 - Numeri naturali 274
- N**
- Nash, equilibrio 618
 - Navigazione dei robot 75
 - Negazione 225
 - Neuroni 14
 - Neuroscienze 13-15
 - Neutralità rispetto al rischio 547
 - Newton-Raphson 126
- O**
- Obiettivo comune 610
 - Obiettivo 56
 - Offerenti 643
 - Offerta 642

- Oggetti 260, 334
---composti 328
---mentali, e logica modale 335-338
---garantiti 527
Oligopolio di Cournot 632
Omeostatici, dispositivi 19
Omogeneo rispetto al tempo 473
Onniscienza 43
---logica 337
Ontologia 279
---superiore 324
Operatore do 462
Optogenetica 15
OR rumoroso 432
Ordinabilità 541
Ordinamento
---dei congiunti 297
---delle mosse 159
---di variabili e valori 198
---topologico 205, 427
Orizzonte
---finito 576
---infinito 576
Optimalità rispetto al costo 80
Ottimamente efficiente 95
Ottimismo in condizioni di incertezza 145
Ottimizzazione
---convessa 127
---vincolata 126
- P**
P 660
Paradosso di Condorcet 648
Parametro 424
Paramodulazione 315
Pareto ottimalità 619
Particle filtering 502
---di Rao-Blackwell 507
Partizione 327
Pattern matching 297
Payoff incerti, e giochi di assistenza 633
PDDL 354
PEAS 45
Penalità 60
Pensare razionalmente: l'approccio delle "leggi del pensiero" 5-6
Pensare umanamente: l'approccio della modellazione cognitiva 5
Percezione 40
Pesatura di verosimiglianza 450
Pianificazione 56
---automatica 353-393
---classica 354
----altri approcci 361
----definizione 354-358
---come soddisfacibilità booleana 360
---con ordinamento parziale 361
---con più agenti, cooperazione e coordinamento 614
---condizionale 380
---decentralizzata 610
---e azione, in ambienti non deterministici 374-384
---e scheduling autonomo 31
---gerarchica 366-374
---Monte Carlo 583
---multiagente 611
---multiattuatore 610
---multibody 610
---online 381
---senza sensori 376
Piano
---condizionale 127
---congiunto 613
---dominato 603
Pigrizia 398
Plateau 118
Playout 167
Polialbero 444
Politica 575
---di selezione 167
---di simulazione 167
---dominante 598
---iterazione 587
----asincrona 588
---miglioramento 587
---non stazionaria 576
---ottima 575
----e utilità degli stati 578

- perdita 586
- propria 577
- quasi ottima per problemi dei banditi 595
- stazionaria 576
- valutazione 587
- POMCP 604
- POMDP
 - algoritmi 601-605
 - online 604
 - definizione 599
 - iterazione dei valori 601
- Portafoglio 388
- Positivismo logico 9
- Posizione 152
- Potatura 95, 152
 - alfa-beta 157
 - in avanti 165
 - indipendente dal dominio 363
- Precalcolo 107
- Precondizione 354
 - mancante 381
- Predizione 476
- Preferenze 399
 - di modelli 343
 - ignote 564-567
 - monotone 545
 - per le clausole unitarie 316
 - razionali, e utilità 543
 - stazionarie 577
- Premessa 225
- Premio
 - di assicurazione 547
 - di esplorazione 595
- Preservazione della verità 223
- Principio
 - di inclusione-esclusione 406
 - di rivelazione 645
- Probabilità 6, 11
 - a posteriori 402
 - a priori 402
 - condizionata 402
 - di accettazione 458
 - marginale 408
 - non condizionata 402
- Probit 435
- Problemi 69
 - conformante 131
 - dei banditi 591-598
 - di Bernoulli 595
 - del commesso viaggiatore 74
 - del mondo reale 71
 - del frame 246
 - della qualificazione 248
 - della ruota di scorta 356
 - di allineamento dei valori 7
 - di coordinamento 610
 - di costruzione di mappe 140
 - di decisione sequenziali 573-583
 - di ottimizzazione 116
 - di vincoli 191
 - di partizione di un insieme 641
 - di rappresentazione del frame 246
 - di ricerca online 140
 - di selezione 597
 - di soddisfacimento di vincoli 185-212
 - definizione 186
 - di viaggio 74
 - e soluzioni 69
 - esemplificativi 71-75
 - formulazione 68-70
 - inferenziale del frame 247
 - rappresentazione con una rete di decisione 556
 - reali 73
 - rilassati 104
 - risoluzione con la ricerca 67-113
 - SAT casuali 243
 - senza sensori 131
 - set-cover 363
 - standardizzati 71
 - struttura 204-208
 - su griglia 71
- Procedura di inferenza 227
- Processo
 - decisionale di Markov 575
 - di Markov 473
 - del primo ordine 473
 - di ricompensa di Markov 592
- Prodotto puntuale 441
- Profilo di strategie 616
- Profondità 80
 - effettiva 103

- Progettazione
---di agenti 611
---di meccanismi 611
---di proteine 75
- Programma
---agente 40, 50-52
---come modello probabilistico 530-534
----funzionamento dei componenti 60-62
---generativo 530
- Programmazione
---di lavori 187
---dinamica 305, 575
----in tempo reale 590
---genetica 121
---lineare 127, 589
---logica 302
----a vincoli 306
----con tabelle 305
---probabilistica 511-538
- Prolog 302
- Propagazione
---degli estremi 194
---dei vincoli 191-196
---delle unità 240
- Propensione al rischio 547
- Proposizionalizzazione 289
- Proprietà 260
---del valore dell'informazione 561
---di raffinamento discendente 369
- Protocollo di concessione monotona 653
- Pseudocodice 668
- Psicologia 15
---cognitiva 16
---evoluzionista 551
- Punto
---di crossover 121
---di riferimento 107
---focale 619
- Q**
- QALY 545
- Quadrato, completamento 491
- Quantificare l'incertezza 397-422
- Quantificatore 267
---esistenziale 268
---universale 267
- Query, nella logica del primo ordine 272
- Quiescenza 164
- R**
- Raccolta di informazioni non miope 562
- Raffinamento 366
- Ragionamento
---automatico 4
---basato sugli obiettivi 239
---con informazione di default 342-346
---guidato dai dati 239
---probabilistico 27-28, 423-470
----nel tempo 471-509
- Random walk 143
- Randomizzazione 54
- Rapporto di competitività 141
- Rappresentazione
---atomica 60
---della conoscenza 4, 323-351
----in un dominio incerto 423
---di MDP 581
---di vincoli temporali e di risorse 384
---distribuita 62
---fattorizzata 61
---locale 62
---strutturata 61
- Rating 516
- Razionalità 4, 42-45
---individuale 637
---limitata 7
- Re Mida, problema 36
- Refutazione 230
- Regola 225
---condizione-azione 52
---del prodotto 402
---della catena 427
---di Bayes 411
----combinazione di evidenze 413
----utilizzo 411-414
---di default 343
---di inferenza 230

- Regressione 359
- Reificazione 326
- Relazione 186, 260
 - di accessibilità 336
- Reti
 - bayesiane 28, 424
 - ibride 433
 - con variabili continue 433
 - dinamiche 496-507
 - inferenza approssimata 447-460
 - inferenza esatta 438-446
 - semantica 426-438
 - causali 461-464
 - di contributi marginali 639
 - di decisione 556-558
 - dinamica 581
 - gerarchiche 366
 - probabilistiche qualitative 554
 - neurali 26
 - semantiche 338
- Reticolo di sussunzione 293
- Revisione delle credenze 344
- Riassumere l'incertezza 398
- Ricerca 56, 69
 - a costo illimitato 96
 - a costo limitato 96
 - a costo uniforme 82
 - a profondità limitata 85
 - a subottimalità limitata 96
 - ad albero 79
 - Monte Carlo 167-170
 - ad approfondimento iterativo 85-86, 97
 - alfa-beta euristica 161-166
 - all'indietro per pianificazione 359
 - beam 97, 111
 - stocastica 121
 - best-first 77
 - greedy o “golosa” 88
 - ricorsiva 98
 - bidirezionale 87
 - con avversari e giochi 151-184
 - con azioni non deterministiche 127-131
 - con backtracking 85
 - per CSP 196-202
 - con memoria limitata 97
 - con osservazioni parziali 131-140
 - con regressione 359
 - di quiescenza 164
 - di soluzioni astratte 369
 - di soluzioni primitive 368
 - euristica bidirezionale 101
 - front-to-end 101
 - front-to-front 101
 - hill climbing 116
 - in ambienti complessi 115-150
 - in ambienti parzialmente osservabili 135
 - in ampiezza 81
 - in assenza di osservazioni 131
 - in avanti
 - gerarchica 374
 - nello spazio degli stati 358
 - in profondità 83
 - e problema della memoria 83
 - incrementale 146
 - informata 88
 - local beam 120
 - locale 116
 - e problemi di ottimizzazione 116-123
 - greedy 117
 - in spazi continui 124-127
 - online 143
 - per CSP 202-204
 - minimax 154
 - Monte Carlo pura 167
 - offline 140
 - online 140
 - operativa 13
 - pesata 95
 - retrograda 166
 - soddisfacente 95
 - su albero, e ricerca in tabelle 166
 - su grafo 79
 - taglio
 - veloce 96
- Ricombinazione 121
- Ricompensa 60, 574
 - media 578
 - puramente additiva 577
 - scontata additiva 576
- Riconoscimento
 - del piano 615
 - vocale 32
- Ridenominazione 295

- Riduzione 444
---all'inferenza proposizionale 288
---delle ultime mosse 165
---simmetrica 364
- Risoluzione 233, 306-317
---binaria 309
---come regola di inferenza 308
---di input 316
---di problemi
----con la ricerca 67-113
----di scheduling 385
----parzialmente osservabili 137
---lineare 316
---unitaria 233
- Risolvente 232
- Risultato sociale 648
- Riusabile 384
- Roba 331
- Robotica 4
- Rollout 167
- Rompicapo
---a 15 tasselli 72
---a 8 tasselli 72
---a tasselli mobili 72
---sokoban 72
- S**
- SAT 230
- Saturazione 312
- Scacchi parzialmente osservabili 174
- Scacco matto
---accidentale 176
---garantito 174
---probabilistico 175
- Scala di ricompensa 580
- Schedule 386
- Scheduling 384
- Schema 123
---di azione 354
---di percezione 375
- Scienze cognitive 5
- Scomponibilità 542
- Scomposizione 365
---ad albero 207
---esaustiva 327
---gerarchica 366
- Scorciatoia 108
- Selezione 121
- Semantica 221, 225
---dei database 271
----di Prolog 305
---della logica del primo ordine 262-272
---delle reti bayesiane 426-438
- Sensore 46
- Separatore 76
- Separazione 414
- Sequenza
---di montaggio automatico 75
---percettiva 40
---più probabile 481
- Sfruttamento di stati 167
- Sillogismi 5
- Simboli
---di costante 264
---di funzione 264
---di predicato 264
---e interpretazioni 264
---di uguaglianza 270
---proposizionali 224
---puro 240
- Simmetria di valore 208
- Simulated annealing 119
- Simulazione 167
- Sincronizzazione 613
- Singolarità 15
- Singolarmente connesse, reti 444
- Sintassi 221, 224
---della logica del primo ordine 262-272
- Sintesi 317
- Sistema
---ad anello aperto 69
---ad anello chiuso 69
---di mantenimento della verità 344-345
---di produzioni 299
---di ragionamento per categorie 338
---esperto 24-26
---fisico di simboli 21

- Skolemizzazione 308
 - SMA 100
 - Smoothing 476, 479
 - a ritardo fisso 481
 - Soddisfacibilità 230
 - Soddisfacimento di vincoli 185-212
 - Soddisfazione 13, 221
 - Softbot 46
 - Soluzione 186
 - ciclica 130
 - di un problema 69
 - ottima 70
 - parziale 186
 - Sostantivo
 - non numerabile 331
 - numerabile 331
 - Sostituibilità 542
 - Sostituzione 272
 - Sotto-obiettivi serializzabili 364
 - Sotto-vincolato 243
 - Sottocategoria 326
 - Sottografo ancestrale 431
 - Sottoproblemi 106
 - indipendenti 204
 - Sovrascrittura 340
 - Spalla 118
 - Sparso 428
 - Spazio
 - campionario 401
 - degli stati 69
 - a livello degli oggetti 108
 - di metalivello 108
 - panorama 116
 - Sperimentazione psicologica 5
 - Spiegazione 346
 - Stack 78
 - Standardizzazione separata 291
 - Stato 69, 354
 - corrente del mondo 245
 - credenza 127, 475
 - e osservazioni 472
 - esplorabile in modo sicuro 141
 - iniziale 69
 - interno 54
 - obiettivo 69
 - ripetuto 79
 - terminale 153
 - Statistica 11
 - d'ordine 548
 - Stima
 - non distorta 547
 - dello stato 138, 475
 - con la logica 248
 - Strategia 616
 - di ricerca informata o euristica 88-102
 - di ricerca non informata 81-88
 - di risoluzione 315
 - di Tipo A 161
 - di Tipo B 161
 - di Zeuthen 653
 - dominante 617, 644
 - evolutiva 121
 - mista 616
 - nei giochi cooperativi 637
 - pura 616
 - Strato 154
 - Struttura
 - dati per la ricerca 77
 - delle preferenze
 - e utilità multiattributo 554
 - di coalizioni 636
 - e risultati 636
 - Sudoku 195
 - Superadditività 636
 - Superintelligenza artificiale 35
 - Superprocesso dei banditi 597
 - Sussunzione 316, 341
- T**
- Tabella
 - delle probabilità condizionate 425
 - delle trasposizioni 161
 - Tagliare la ricerca 163
 - Tasso di mutazione 121
 - Tautologia 230
 - Tavola di verità 226
 - Tecnica di Rao-Blackwell 507

- Tempo 333
---e incertezza 472-475
---di arresto 593
---di mixing 478
---discreto 472
---scheduling e risorse 384-387
Tenere traccia di un mondo complesso 525-530
Teorema 274
---del limite centrale 664
---dell'equivalenza dei ricavi 646
---dello shaping 580
---di deduzione 230
---di Gibbard-Satterthwaite 650
---di Herbrand 312
---di incompletezza 11
---di risoluzione ground 236
Teoria
---dei giochi 152-154, 610
----cooperativi 635-642
----non cooperativi 615-635
---del controllo 18
----e cibernetica 18-19
---del valore dell'informazione 558
---dell'utilità 399
----basi 541-544
----multiatributo 551
---della conferma 9
---della probabilità 399
----assiomi 405
----notazione base 401-407
---della scelta sociale 648
---delle decisioni 12, 400
---descrittiva 549
---normativa 549
Terminazione anticipata della simulazione 170
Termini 266
---ground 267
Test
---di taglio 161
---di terminazione 153
---di Turing 4
----totale 4
---obiettivo
----a posteriori 81
----anticipato 81
Testa 237
- Time slice 472
Tit-for-tat 624
Tracciamento multitarget 526
Tracciato di esecuzione 530
Tradizione logicista 6
Traduzione automatica 32
Tragedia dei beni comuni 647
Transitività 541
Trashing 100
Trasparenza referenziale 336
Trasporto aereo di merci 355
Trasposizione 161
Trattabilità 12
Tupla 263
- U**
- UCB1 168
UCT 168
Uguaglianza 270, 314
---simbolo 270
Unificatore 291
---più generale 291
Unificazione 291
---e inferenza del primo ordine 289-294
---equazionale 315
Universo di Herbrand 312
Unrolling 518
Utilità 10, 57
---attesa 57, 540
----e delusione dopo la decisione 547
---del denaro 545
---nel tempo 576
---normalizzate 544
Utilitarismo 10
- V**
- Validità 230
Valore 61
---atteso 162, 172, 664
---del materiale 162
---dell'informazione perfetta 559

- dell'informazione 558-564
- di backup 98
- di default 340
- di Shapley 638
- di una vita statistica 545
- di verità 225
- expectiminimax 172
- meno vincolante 199
- minimax 154
- monetario atteso 546
- Valutazione**
 - del livello di abilità dei giocatori 516
 - delle reti di decisione 558
 - e scale di utilità 544
- Variabile** 61, 125, 267
 - atemporale 245
 - casuale 403
 - di base 514
 - nascosta 436
 - non modellata 462
 - numero 520
 - Varianti**
 - del formalismo CSP 188
 - non indicizzabili 596
 - Varianza** 664
 - Veicoli robotizzati** 31
 - Velocità di mixing** 457
 - Verifica** 317
 - dei circuiti 283
 - in avanti 199
 - Verità** 221
 - preservazione 223
 - Vettori** 661-662
 - dei payoff 636
- Vickrey–Clarke–Groves** 647
- Vicolo cieco** 141
- Vincolo**
 - di azione concorrente 614
 - di precedenza 188
 - di preferenza 190
 - di risorse 384
 - di rottura della simmetria 208
 - disgiuntivo 188
 - globale 189, 194
 - lineare 189
 - non lineare 189
 - su preferenze razionali 541
 - sulle risorse 194
 - unario 189
- Visione artificiale** 4
- Votazione** 648
- Voto**
 - a maggioranza relativa 649
 - a maggioranza semplice 649
 - alternativo 649
 - maggioritario 650
 - per approvazione 649
- W**
- Wumpus, mondo** 218-221
- Z**
- Zucchero sintattico** 275

